Weltner · Hornig

AMIGA



TIPS & TRICKS

Ein **DATA BECKER** Buch

Weltner · Hornig

AMIGA



TIPS & TRICKS

Ein **DATA BECKER** Buch

ISBN 3-89011-211-0

Vorwort

Schauen Sie einmal auf den Umschlag dieses Buches. Dort finden Sie zwei Namen. Doch damit sind noch lange nicht alle Personen genannt, die dazu beigetragen haben, daß dieses Buch jetzt vor Ihnen liegen kann.

Darum danken wir Dave und Carol Mansfield in Kalifornien/USA sowie Lotta Malmhester in Schweden für die Stärkung unserer Arbeitsmoral, den EPSON-Leuten für die zwei Drucker die wirklich hervorragend sind, Herrn Spille von Hewlett Packard für seine Nachsicht, als wir den HP Laserjet nicht zurückgeben wollten, Herrn Schmitz von Commodore-Amiga, der sich bereitwillig befragen ließ, Oda und Sonka für die Zuführung der notwendigen organischen Energien, unseren Eltern für die Geduld im Umgang mit uns, und schließlich unseren gesamten Freunden, weil sie großzügigerweise diesmal keine einzige Diskette löschten.

Wir danken jedoch nicht Gonzo, der uns durch zahlreiche Partys unablässig von der Arbeit ablenken wollte - und das auch geschafft hat.

Rinteln, im Januar 1986

Ralf Hornig *Tobias Weltner*

Inhaltsverzeichnis

1. Einleitung

Vielleicht ergeht es Ihnen ähnlich wie uns, als wir das erste Mal mit dem Amiga konfrontiert wurden: Staunen, Verwunderung, Verblüffung. Ja, dies ist wirklich eine Supermaschine - doch unter der Flut von Möglichkeiten drohte unser Programmiereifer im Keime zu ersticken. Wo soll man denn beim Amiga anfangen; wie ihn jemals verstehen?

Mittlerweile haben wir uns mit dem Amiga mehrere Monate beschäftigt, und das Ergebnis liegt vor Ihnen: sie finden eine Reihe von Programmen, die Sie leicht verständlich Stück für Stück die Fähigkeiten des Amigas erschließen lassen.

Der erste Teil dieses Buches arbeitet mit Microsoft's AmigaBASIC und seiner Programmierung. Das Hauptgewicht wurde auf die Nutzung der bestehenden System-Software-Module gelegt, und so finden Sie Routinen zur Nutzung der verschiedenen Schriftarten, eine Grafikbildschirm-Hardcopy-Routine, einen Diskmonitor usw.

Im zweiten Teil kommt "C" zum Zuge. Anhand vieler Beispiele werden die Grundzüge der Amiga-Systemarchitektur nicht nur theoretisch erklärt, sondern auch durch funktionsfähige Programme belegt. So dürfte es für Sie nicht lange dauern, mit der Programmierung der Menüs, Windows und Requesters vertraut zu werden. Und wenn all dies nichts Neues für Sie ist, wie wär's dann mit Programmen zum neuen Half-Brite-Modus, mit dem sich 64 Farben gleichzeitig darstellen lassen und der erst nach Drucklegung der Amiga-Handbücher implementiert wurde?

15 Fragen zum Amiga

Nach einigen Gesprächen mit frischgebackenen Amiga-Besitzern stellten wir fest, daß einige Fragen immer wieder auftauchen. Da wir davon ausgehen, daß auch viele von Ihnen sich diese Fragen stellen werden, geben wir Ihnen hier Antwort.

Frage 1: Was ist das CLI?

Antwort: CLI steht für *Command Line Interface*. Es ist eine Benutzeroberfläche, die sich ohne die vielen bunten Symbole (Icons) und ohne Maus ganz auf die Tastatur verläßt. Das CLI kennt im Moment ca. 50 verschiedene Befehle (Workbench Version 1.2), es werden aber in Zukunft noch viele weitere dazu kommen.

Das CLI arbeitet eng mit AmigaDOS, dem Disk Operating System, zusammen. Viele Spezialbefehle erleichtern den Umgang mit Disketten; einige Funktionen lassen sich sogar mit Intuition Fenster- und Maus-Technik überhaupt nicht lösen.

Das CLI wird normalerweise vom Benutzer via Intuition aufgerufen. Es ist aber auch möglich, beliebige CLI-Kommandos von BASIC-Programmen aus zu benutzen (dasselbe gilt für "C").

Frage 2: Wie gelange ich zum CLI?

Antwort: Das Command Line Interface befindet sich serienmäßig auf jeder Workbench-Diskette. Es gibt eine Reihe von Wegen, um an's CLI heranzukommen:

a) Mit *Intuition*. Dies ist der Normalfall. Sie haben Ihr System gebootet, die Workbench-Diskette befindet sich im Laufwerk. Vor Ihnen liegt nun der tiefblaue "Workbench-Screen". Folgen Sie diesen Schritten:

- *Klicken Sie den Disk-Icon der Workbench-Disk an.*

  Es wird sich ein Fenster namens "Workbench..." mit allerlei Dingen darin öffnen.

- *Klicken Sie die Schublade "System" an.*

Es wird sich ein Fenster namens "System" eröffnen, wiederum randvoll gefüllt mit den Errungenschaften unserer heutigen Leistungsgesellschaft, vor allem aber mit einem Icon namens "CLI", das entweder "1)" darstellt (Version 1.1) oder ein nettes kleines Fenster mit 1) darin (Version 1.2 und darüber).

Sollten Sie KEIN CLI-Icon vorfinden, so liegt das daran, daß bei Ihrer Workbench-Diskette noch die Kindersicherung eingeschaltet war. Klicken Sie in diesem Fall das Icon "Preferences" im "Workbench..."-Fenster an, und schalten Sie im Feld CLI von OFF auf ON. Saven Sie am besten das Ergebnis mit SAVE gleich wieder ab. Nun müssen Sie das "System"-Fenster schließen und erneut öffnen. Und siehe da: Ein CLI-Icon!

- *Klicken Sie das CLI-Icon an.*

Es wird unverzüglich ein Fenster namens "New CLI" erscheinen, das Sie in alle Richtungen vergrößern und -kleinern können, das jedoch keinen Ausknips-Knopf in der linken oberen Ecke besitzt. Sie haben es geschafft! Sie haben Ihr eigenes CLI!

b) Mit *AmigaDOS*. AmigaDOS verfügt über den Befehl "Execute", mit dem beliebige CLI-Befehle ausgeführt werden können. AmigaDOS wiederum kann über die systeminternen Libraries angesprochen werden. Auf diese Art und Weise kommen selbst AmigaBASIC und "C" ans CLI (siehe Beispielprogramme in diesem Buch).

c) Noch raffinierter! Ein ganz einfacher Weg ist der folgende: Sie haben gerade Ihr System eingeschaltet. Die Kickstart-Disk ist bereits erfolgreich geladen worden, nun flimmert eine gigantische Hand mit einer Workbench-Disk darin auf dem Bildschirm. Bis jetzt ist alles Routine. Aber jetzt! Legen Sie die Workbench-Disk ins Laufwerk. Die Hand

verschwindet, das System bootet. Geben Sie nun rasch "Tobi ist lieb!" über die Tastatur ein! Nun brauchen Sie nur noch zu warten, bis das Laufwerk zum Stillstand gekommen ist. Ist die rote Lampe erloschen, dann nehmen Sie die Workbench-Disk rasch aus dem Laufwerk. Löschen Sie nun "Tobi ist lieb!" wieder mit der BACKSPACE-Taste. Sobald der letzte Buchstabe getilgt ist, treten eine Reihe Fehlerrequester auf. Beachten Sie sie einfach nicht, sondern klicken Sie resolut immer wieder auf CANCEL. Endlich erscheint ein

1)

auf dem Bildschirm - das CLI! Geben Sie schnell noch

1) loadwb

ein, und das CLI steht zu Ihrer vollen Verfügung!

Frage 3: Wie kriege ich das CLI wieder weg?

Antwort: Das CLI-Fenster hat keinen Ausschalter. Es löst sich mit folgender Eingabe von selbst in Wohlgefallen auf:

1) endcli

Haben Sie von einem CLI aus Programme gestartet, dann bleibt das CLI-Fenster allerdings solange erhalten, wie die Programme laufen.

Frage 4: Ich habe keine Schreibmaschine, aber einen an meinen Amiga angeschlossenen Drucker. Kann man da nichts machen?

Antwort: Na klar! Das CLI ist der geborene Problemlöser. Geben Sie folgendes CLI-Kommando ein:

*1) copy * to prt:*

wobei * das Symbol des aktiven CLI-Fensters ist. Nach dieser Eingabe verschwindet der gewohnte CLI-Prompt "1)", lediglich der Cursor hält Ihnen die Treue. Jede Tastatureingabe wird nun nach Druck der RETURN-Taste auf den Drucker ausgegeben - quasi eine Schreibmaschine mit einzeiligem Korrekturspeicher!

Aus dem Schreibmaschinen-Modus kommen Sie wieder heraus, wenn Sie gleichzeitig die CTRL- und \-Taste drücken.

Je nach Lust und Laune können Sie übrigens auch Text in ein anderes Fenster kopieren...

*1) copy * to CON:10/10/300/100/Kopie-Text*

...oder sich selbst wiederholen...

*1) copy * to **

Frage 5: Ich besitze nur ein Diskettenlaufwerk. Jedes Mal, wenn ich einen CLI-Befehl verwende, muß ich kurz die Workbench-Diskette einlegen. Kann man das nicht verhindern?

Antwort: Jedes CLI-Kommando ist auf der Workbench-Disk als kleines Programm im Directory "c:" abgespeichert. Wenn Sie nun einen CLI-Befehl verwenden, lädt Amiga diesen normalerweise jedesmal von der Workbench-Disk nach. Dadurch spart man natürlich eine Menge Speicherplatz, denn die CLI-Kommandos belegen so keinen Systemspeicher. Auf der anderen Seite muß man laufend Disketten wechseln, wenn man nur über ein Laufwerk verfügt.

Wenn Sie über genügend Speicher verfügen, können Sie aber alle (oder selektierte) CLI-Befehle ins RAM kopieren. Das geht so:

1) makedir ram:c
1) copy sys:c to ram:c
1) assign c: ram:c

Zunächst wird in der RAM-Disk ein Unterdirectory namens "c" angelegt. Anschließend werden alle CLI-Befehle in dieses Directory kopiert. Schließlich wird Amiga mitgeteilt, daß das Kommando-Directory c: nun auf der RAM-Disk liegt.

Ist Ihnen der Speicher Ihres Rechners doch noch zu schade, dann können Sie sich auf die meistbenutzten CLI-Kommandos beschränken. Zum Beispiel so:

1) makedir ram:c
1) copy sys:c/copy to ram:c
1) copy sys:c/dir to ram:c
1) copy sys:c/list to ram:c
(...)
1) assign c: ram:c

Wollen Sie wieder zurück zur Workbench-CLI, dann funktioniert das in jedem Fall (sofern Sie die Workbench-Disk in Laufwerk 0, das eingebaute, legen):

1) df0:c/assign c: df0:c

Nach Beendigung Ihrer Vorhaben empfiehlt es sich, das RAM-CLI wieder zu löschen, um Speicherplatzreserven zurückzubekommen:

1) delete ram:c#?
1) delete ram:c

Frage 6: Gibt es einen Joker, vergleichbar dem * bei alten Commodore-Rechnern?

Antwort: Ja, es handelt sich um die Symbolkombination #?. Das Zeichen * repräsentiert ja bereits das aktuelle CLI-Fenster.

1) delete ram:#?

löscht die gesamte RAM-Disk.

1) run amig#?

funktioniert jedoch beispielsweise nicht, denn Amiga weiß nicht, welches Programm ausgeführt werden soll. Es könnten ja mehrere Programme existieren, die mit der Buchstabenkombination "amig" beginnen.

Frage 7: Ich möchte gern eine Liste aller CLI-Befehlsworte auf den Drucker ausgeben. Geht das?

Antwort: Das funktioniert mit einer einfachen Befehlskombination:

1) list quick sys:c to prt:

Die Option "quick" bewirkt, daß nur die Befehlsnamen ausgegeben werden. Erschaffungsdaten, Uhrzeit, Schutzstatus sowie Filegröße werden nicht ausgedruckt. Die CLI-Befehle selbst stehen im Unterdirectory c: des Systemdirectorys sys:.

Schneller geht es, wenn Sie die Multitasking-Fähigkeiten Ihres Amigas ausnutzen:

1) run list quick sys:c to prt:

Hier wird ein weiterer Task eröffnet, der die Druckerausgabe bewerkstelligt. Sie können also gleich mit anderen Sachen weiterarbeiten, während Ihr Amiga quasi im Hintergrund die Befehlsworte ausgibt.

Frage 8: Gibt es eine Möglichkeit, die Befehlssyntax eines bestimmten CLI-Befehls herauszufinden?

Antwort: Fast alle CLI-Befehle verfügen über eine Eingabehilfe. Falls Sie also nicht mehr genau wissen, wie ein spezieller Befehl aufgerufen wird, dann geben Sie einfach den Befehlsnamen, gefolgt von einem Leer- und einem Fragezeichen ein. Zum Beispiel:

1) list ?

Das Ergebnis ist:

DIR,P=PATH/K,KEYS/S,DATES/S,NODATES/S,TO/K,S/K, SINCE/K, UPTO/K,QUICK/S:

Na, alles klar?

DIR steht für ein Directory, kann aber auch weggelassen werden (dann wird das augenblickliche Directory ausgegeben). Alle weiteren Angaben enthalten neben dem Optionswort eine Bedingung:

/A: Dieses Argument muß angegeben werden.

/K: Dieses Argument muß in Verbindung mit einem Parameter gegeben werden.

/S: Dieses Argument steht für sich allein.

So sind die folgenden Kommandos möglich:

1) list df0: keys nodates

Gibt das Inhaltsverzeichnis "df0:" mit den jeweiligen Anfangsblöcken, jedoch ohne Datumsangabe aus.

1) list df0: since 04-Oct-86 upto today

Gibt die Programme des Inhaltsverzeichnisses "df0:" aus, die zwischen dem 4. Oktober 1986 und heute geschrieben worden sind.

Frage 9: Wie lassen sich CLI-Kommandos unterbrechen?

Antwort: CTRL-C unterbricht einen Befehl. CTRL-D veranlaßt einen Execute-Befehl, so schnell wie möglich den Programmab-

lauf zu unterbrechen. CTRL-E und CTRL-F werden nur in ganz besonderen Fällen gebraucht.

Frage 10: Ich verfüge über ein Laufwerk und möchte ein Programm kopieren. Wie funktioniert das?

Antwort: Erste Möglichkeit: Es handelt sich um ein kleines Programm. Laden Sie es zunächst in die RAM-Disk:

1) copy programm to ram:
1) copy c/copy to ram:

Der Copy-Befehl wurde in der zweiten Anweisung ebenfalls kopiert, um zu verhindern, daß die Workbench nachgelegt werden muß. Legen Sie nun die Zieldiskette ins Laufwerk. Anschließend wird das Programm zurückkopiert:

1) ram:copy ram:programm to df0:

Legen Sie nun bitte wieder die Workbench-Disk ein. Die RAM-Disk muß nur noch gelöscht werden, und schon sind wir fertig:

1) delete ram:#?

Eine andere Methode bedient sich der Intuition-Icons. Sie müssen dazu zunächst die Originaldiskette einlegen und den Disk-Icon anklicken. Sobald das Icon des gewünschten Programms erscheint, legen Sie die Zieldiskette ein. Öffnen Sie auch diese durch Anklicken des Disk-Icons. Nun können Sie das Icon des Originalprogramms mit Hilfe der Maus ins Fenster der Zieldiskette bewegen.

Der Rest geschieht automatisch per Requester: Die Disketten müssen ein paarmal gewechselt werden.

Achtung: Es gibt Programme, die gar kein Icon besitzen. Sie erscheinen also auch gar nicht als Symbol in einem Intuition-Fen-

ster. Sie können aber solch einem Programm ein Icon beschaffen! Legen Sie dazu die Workbench-Disk ein. Die folgenden Zeilen sind nötig:

1) copy df0:clock.info to ram:
1) rename ram:clock.info as ram:programm.info
1) copy c/copy to ram:

Jetzt legen Sie die Diskette ein, auf der sich Ihr Originalprogramm befindet. Geben Sie nun ein:

1) ram:copy ram:programm.info to df0:

Nun hat Ihr Programm (hier namens programm) ein Icon. Legen Sie wieder die Workbench ein und löschen Sie die RAM-Disk:

1) delete ram:#?

Frage 11: Ich habe zwei Laufwerke und möchte ein Programm kopieren. Ein leichtes Unterfangen?

Antwort: Sicher. Es genügt eine CLI-Zeile:

1) copy df0:originalprogramm to df1:

Hierbei muß sich das Originalprogramm in Laufwerk 0 im Directory df0: befinden.

Falls Ihr Programm ein Icon besitzt, dann muß auch dieses kopiert werden:

1) copy df0:originalprogramm.info to df1:

Natürlich können Sie auch per Intuition das Programm-Icon direkt von einer Diskette zur anderen schieben (siehe Frage 10).

Frage 12: Ich möchte eine ganze Diskette kopieren. Wie geht das?

Antwort: Mit dem Befehl "diskcopy". Es spielt dabei keine Rolle, ob Sie über ein- oder mehrere Drives verfügen. Achtung: Um ungewolltes Löschen der Original-Disk auf jeden Fall zu verhindern, sollten Sie den Schutzpin an der Seite der Originaldiskette zumindest für die Dauer des Kopierens nach oben schieben, falls dies noch nicht getan wurde.

Sie besitzen EIN Laufwerk:

1. Legen Sie die Workbench-Diskette ein.
2. Tippen Sie den CLI-Befehl ein:

1) diskcopy from df0: to df0: name "kopie"

Nun erscheint die Aufforderung, die Quell-Diskette (SOURCE) einzulegen. Kommen Sie dem nach. Nach einer Weile muß die Ziel-Diskette (DESTINATION) eingelegt werden, und nach ein paar weiteren Wechseln haben Sie es geschafft.

Sie besitzen ZWEI Laufwerke:

1. Legen Sie die Workbench-Diskette ein.
2. Tippen Sie den CLI-Befehl ein:

1) diskcopy from df0: to df1: name "kopie"

Stecken Sie nun gemäß der Aufforderung die Quell-Disk in Laufwerk 0, die Ziel-Disk in Laufwerk 1. Die Disketten brauchen natürlich nicht mehr gewechselt werden.

Frage 13: Was ist eine Startup-Sequence und was kann man mit ihr machen?

Antwort: Die Startup-Sequence ist eine Liste von CLI-Befehlen, die ganz zu Anfang beim Booten des Systems ausgeführt wird. Wie das aussieht, können Sie sich leicht veranschaulichen:

1) execute s/startup-sequence

Sie können sich diese CLI-Befehle auch einmal anschauen:

1) type s/startup-sequence

Wenn Sie Lust haben, können Sie sich auch eine eigene Startup-Sequenz schreiben. Sie sollten jedoch beachten, daß das Kommando loadwb unbedingt übernommen wird, da sonst das Intuition-Icon-System nicht aktiviert wird. Sollten Sie dann einmal die CLI durch endcli verlassen, stehen Sie quasi vor zugeschlagener Haustür, denn es wären keine Icons da, nur ein leerer Bildschirm.

Eine eigene Startup-Sequenz kann man über das Kommando "ed" erstellen:

1) ed s/startup-sequence

Bei Version 1.2 der Workbench sehen Sie nun:

```
echo "Workbench Diskette (Version 1.2/33.43)"
echo " "
echo "(Datum und Uhrzeit mit 'Preferences' einstellbar)"
if EXISTS sys:system
     path sys:system add
endif
BindDrivers
setmap d
LoadWb
endcli > nil:
```

Mit den Cursortasten können Sie den Cursor bewegen. Auf Druck der ESC-Taste landen Sie in der Kontrollzeile. Ein "d" löscht die Zeile, in der sich der Cursor zuletzt befand. Löschen Sie beispielsweise die Anweisungen.

setmap d
endcli) nil:

Nun saven Sie die Sequenz wieder durch Druck auf die ESC-Taste und anschließend "x".

Probieren Sie die neue Sequenz doch gleich mal aus!

1) execute s-startup-sequence

Wie Sie sehen, haben Sie wieder eine amerikanische Tastaturbelegung, und das CLI-Fenster ist auch nicht verschwunden.

Frage 14: Kann der Amiga im CLI sprechen?

Antwort: Sicherlich, wenngleich man auch keine Parameter verändern kann. Das Kommando heißt "say" und wirkt wie ein Print-Kommando. Leider ist es unmöglich, Programm-Files aussprechen zu lassen. Lediglich unmittelbar nachfolgender Text wird verlesen:

1) say tobi is a real nice guy!

Interessant ist dieser Befehl auch in Zusammenhang mit der Startup-Sequenz (Frage 13)! Stellen Sie sich vor, Ihr Amiga begrüßt Sie jedesmal nach dem Einschalten mit einem netten "Guten Tag!".

Frage 15: Wie kann ich ein C-Listing auf den Drucker ausgeben?

Antwort: Am besten geschieht dies mit dem CLI-Befehl type. Ein Beispiel: Sie haben ein C-Listing, erstellt wahrscheinlich mit ed unter dem Namen test.c auf df1:. Geben Sie nun ein:

run type df1:test.c to prt: opt n

Mit "run" nutzen Sie die Multitasking-Fähigkeit des Amigas - während der Drucker lustig rattert, können Sie schon wieder etwas anderes fabrizieren. Der Zusatz "opt n" sorgt dafür, daß der Ausdruck des C-Listings mit Zeilennummern versehen wird. Dies ist recht hilfreich bei der Fehlersuche.

2. Das AmigaBASIC

BASIC (Beginners All Purpose Symbolic Instruction Code) war ursprünglich ein Produkt der Verzweiflung. Es geschah in der Zeit, als Computerprogramme noch in mühsamer Kleinarbeit assembliert werden mußten und Compiler auch keine Alternative für Anfänger darstellten, da man nach jedem Programmierfehler von vorn beginnen mußte. Ein findiger Mensch am Dartmouth College dachte etwas nach und entwickelte eine besonders "anfängerfreundliche" Sprache, die einen aus englischen Worten bestehenden Befehlssatz beinhaltete und ohne Compilierung auskam. BASIC war geboren, und das Konzept des "Symbolic Instruction Code" hat sich bewährt: Heute ist BASIC die am häufigsten benutzte Programmiersprache der Welt.

BASIC wurde im Laufe der Jahre erweitert, verbessert, ausgedehnt. Ein fortschrittliches BASIC, wie es das AmigaBASIC ist, verbindet nun die leichte Erlernbarkeit der Befehlsworte mit den Vorzügen der strukturierten Programmierung, wie es das geübte Auge bisher nur von Compilern gewohnt war. BASIC ist "erwachsen" geworden; mühelos kann der einst so gefürchtete Spaghetti-Code umschifft werden und Programme lassen sich auch noch nach Jahren verstehen, nachvollziehen und gegebenenfalls verändern.

AmigaBASIC stammt aus dem Hause Microsoft, und Computerfreaks werden jetzt sicher die Ohren spitzen. Sollte es möglich sein, daß Microsoft (eine Firma, die für jeden nur erdenklichen Computer qualitativ hochwertige BASIC Interpreter vorrätig hat) einen Interpreter speziell für den Amiga und seine Fähigkeiten entwickelt hat? Leider ist dem nicht so. Vielmehr handelt es sich um eine an den Amiga angepaßte Version des Microsoft-BASICs für den Macintosh. Zwar wird Amiga's moderne Window- und Menü-Technik voll unterstützt, aber trotzdem werden lange Gesichter nicht vermieden werden können. Was ist mit dem "Hold-And-Modify"-Modus passiert? Wurden die Disk-Zeichensätze vergessen? Wo sind die ausführlichen Disketten- und Synthesizer-Befehle? Bevor Sie sich jetzt voller Entrüstung abwenden, muß zu Microsofts Ehrenrettung der in dieser Form revolutio-

näre "LIBRARY"-Befehl vorgeschoben werden. Mit seiner Hilfe werden Sie die fehlenden Befehle leicht nachträglich implementieren können. Wie's geht erfahren Sie im zweiten Teil dieses Kapitels.

Bevor wir in die Tiefen des BASIC Interpreters hinabtauchen, wollen wir darauf hinweisen, daß vor Ihnen das Buch "Amiga Tips&Tricks" von DATA BECKER liegt. Bitte erwarten Sie kein langweiliges und -wieriges BASIC-Handbuch. Die folgenden Seiten beschäftigen sich mit speziellen, ungewöhnlichen, revolutionären Programmen und Techniken, die Programmierkenntnis voraussetzen (die ausgedruckten Programme laufen natürlich immer, egal ob sie verstanden worden sind oder nicht). In Zweifelsfällen ziehen Sie bitte das Handbuch für AmigaBASIC oder das AmigaBASIC-Buch von DATA BECKER zu Rate.

2.1 Der Amiga und der Rest der Welt

Weil das AmigaBASIC so ungeheuer flexibel ist, ist es wichtig, sich zunächst einmal mit dem Aufbau des Amiga-Betriebssystems auseinanderzusetzen.

Das Amiga Kernel ist in System Modulen zusammengefaßt, die zusammen eine Hierarchie bilden. Ganz unten steht die Hardware, die letztendlich die Arbeit erledigt. Darüber gibt es verschiedene Interpretationsstufen. Level 2 ist am maschinennächsten und steuert die Hardware. Die Stufe darüber nimmt bereits an Komplexität zu und die Kommandos werden mit zunehmender Stufe immer besser für den Menschen verständlich.

An der Spitze der Pyramide finden Sie die Schnittstelle zum Menschen, am Boden der Hierarchie befindet sich die Schnittstelle zur Maschine. Alle System-Module dazwischen sorgen für die schrittweise Umsetzung der Eingabe in eine für den Amiga verständliche Befehlskette. Die einzelnen Verästelungen sorgen dafür, daß alle Hardware-Komponenten (CPU, Disk, Grafikchips) genau die Anweisungen bekommen, für die sie zuständig sind.

In unserem Fall bietet AmigaBASIC normalerweise die Schnittstelle zum Menschen, denn Sie geben in den Computer BASIC-Befehlswörter ein (wenn Sie zum Beispiel mit dem CLI arbeiten, dann wird dieses die Schnittstelle zum Menschen).

AmigaBASIC läßt jedoch sehr flexible Programmierung zu. Neben der Benutzung der BASIC-Befehle besteht die Möglichkeit, sich in der Hierarchie herabzuarbeiten und System-Module direkt anzusprechen. Je weiter Sie sich herabarbeiten, desto maschinennäher werden die benötigten Anweisungen sein (desto komplizierter also), desto mehr Spielraum werden Sie aber auch haben. So können Sie durch direkten Zugriff auf die System-Module Funktionen programmieren, für die Sie keinen äquivalenten BASIC-Befehl finden würden. Zum Beispiel die Benutzung der Disk-Zeichensätze in BASIC.

Jedes der System-Module besteht aus einer Befehlslibrary. Diese enthält eine Vielzahl von Maschinenroutinen für die verschiedensten Dienste. Der Befehl

```
OpenDiskFont(textAttr)
```

öffnet beispielsweise einen neuen Zeichensatz, vorausgesetzt, die richtigen Informationen werden mitgeliefert.

2.2 Implementierung der Amiga-Kernel-Befehle

Wie gesagt, bei den System-Modul-Kommandos handelt es sich um Maschinensprache-Routinen, die selbst AmigaBASIC nicht ohne weiteres versteht. Für jede Library, die Sie mit Hilfe des "LIBRARY"-Befehls in Ihrem Programm nutzen wollen, muß ein spezielles Definitionsfile existieren (siehe auch AmigaBASIC Handbuch Anhang F). Das Amiga-Betriebssystem verfügt derzeit über 13 Libraries, aber nur 5 sind für BASIC wirklich interessant. Es sind dies:

1. *exec.library*

   Zuständig für Tasks, Listen, I/O, allgemeine Systembelange, Speicherverwaltung

2. *graphics.library*

   Zuständig für Text und "GELs" (graphic elements)

3. *intuition.library*

   Zuständig für Windows, Screens und Requesters, Alarmmeldungen

4. *dos.library*

   Zuständig für elementare Disk-Befehle

5. *diskfont.library*

   Zuständig für die auf Disk gespeicherten Zeichensätze des Amigas

2.2.1 Das Anpassungsfile ".bmap"

Ein Definitionsfile enthält für jeden Befehl, der im System-Modul liegt, die folgenden Informationen:

a) Name des Befehls (in ASCII), mit 0 abgeschlossen.

b) Offset des Befehls in der System-Modul-Tabelle

c) Übergabe-Register

```
1 = Datenregister d0
2 = Datenregister d1
3 = Datenregister d2
4 = Datenregister d3
5 = Datenregister d4
```

```
 6 = Datenregister d5
 7 = Datenregister d6
 8 = Datenregister d7

 9 = Adreßregister a0
10 = Adreßregister a1
11 = Adreßregister a2
12 = Adreßregister a3
13 = Adreßregister a4
```

mit "0" abgeschlossen.

Für die "graphics.library" und "dos.library" gibt es diese Files bereits. Sie sind als "graphics.bmap" und "dos.bmap" im Directory "Basic Demos" auf der "Extras"-Diskette zu finden. Allerdings läßt sich "dos.bmap" größtenteils in BASIC nicht benutzen, da die dort definierten Befehlsnamen mit bereits bestehenden BASIC-Kommandos kollidieren (z.B. Open, Read, etc.).

Nutzen Sie deshalb einfach die nachfolgenden vier BASIC-Loader, die die nötigen ".bmap"-Files generieren. Haben Sie diese erst einmal auf Diskette, dann können die Loader wieder gelöscht werden. Wir raten trotzdem davon ab, da sie zu Kontrollzwecken recht nützlich sind.

```
REM ****************************************
REM * exec     .bmap linker file GENERATOR *
REM ****************************************
REM *  written 6/23/86 by tobias weltner   *
REM *--------------------------------------*
REM ****************************************
REM * (C) 1986 by DATA BECKER GmbH W.-Germ.*
REM ****************************************

Header:

PRINT "[exec.bmap] linker file GENERATOR"


FunctionDefinitions:

DATA InitCode,72
DATA InitStruct,10,11,1,78
DATA MakeLibrary,10,11,12,1,2,84
DATA MakeFunctions,90
DATA FindResident,96
DATA InitResident,102
DATA Alert,108
DATA Debug,114
DATA Disable,120
DATA Enable,126
DATA Forbid,132
DATA Permit,138
DATA SetSR,1,2,144
DATA SuperState,150
DATA UserState,1,156
DATA SetIntVector,1,10,162
DATA AddIntServer,1,10,168
DATA RemIntServer,1,10,174
DATA Cause,10,180
DATA Allocate,10,1,186
DATA Deallocate,10,11,1,192
DATA AllocMem,1,2,198
DATA AllocAbs,204
```

```
DATA FreeMem,10,1,210
DATA AvailMem,2,216
DATA AllocEntry,9,222
DATA FreeEntry,9,228
DATA Insert,9,10,11,234
DATA AddHead,9,10,240
DATA AddTail,9,10,246
DATA Remove,10,252
DATA RemHead,9,258
DATA RemTail,9,264
DATA Enqueue,9,10,270
DATA FindName,9,10,276
DATA AddTask,10,11,12,282
DATA RemTask,10,288
DATA FindTask,10,294
DATA SetTaskPri,10,1,300
DATA SetSignal,1,2,306
DATA SetExcept,1,2,312
DATA Wait,1,318
DATA Signal,10,1,324
DATA AllocSignal,1,330
DATA FreeSignal,1,336
DATA AllocTrap,1,342
DATA FreeTrap,1,348
DATA AddPort,10,354
DATA RemPort,10,360
DATA PutMsg,9,10,366
DATA GetMsg,9,372
DATA ReplyMsg,10,378
DATA WaitPort,9,384
DATA FindPort,10,390
DATA AddLibrary,10,396
DATA RemLibrary,10,402
DATA OpenLibrary,10,1,408
DATA CloseLibrary,10,414
DATA SetFunction,10,9,1,420
DATA SumLibrary,10,426
DATA AddDevice,10,432
DATA RemDevice,10,438
DATA OpenDevice,9,1,10,2,444
```

```
DATA CloseDevice,10,450
DATA DoIO,10,456
DATA SendIO,10,462
DATA CheckIO,10,468
DATA WaitIO,10,474
DATA AbortIO,480
DATA AddResource,10,486
DATA RemResource,10,492
DATA OpenResource,10,498
DATA GetCC,528
DATA end_of_functions

RESTORE FunctionDefinitions

OPEN "exec.bmap" FOR OUTPUT AS 1

ReadLibFunc:
READ Routine$
gencount=gencount+1
LOCATE 3,1
PRINT "PROCESS: Reading       FUNCTION: ";Routine$+SPACE$(2
  0-LEN(Routine$))
IF Routine$="end_of_functions" THEN ShutDown

counter=0

ReadLoop:
READ value(counter)
IF value(counter)<20 THEN
  counter=counter+1
  GOTO ReadLoop
ELSE
  offset=value(counter)
  counter=counter-1
END IF

offset=65536&-offset
off1=INT(offset/256)
off2=offset-(256*off1)
```

```
lib$=Routine$+CHR$(0)+CHR$(off1)+CHR$(off2)

FOR loop=0 TO counter
  lib$=lib$+CHR$(value(loop))
NEXT loop

lib$=lib$+CHR$(0)

LOCATE 4,1
PRINT "PROCESS: writing        #:";gencount

ges$=ges$+lib$

GOTO ReadLibFunc

ShutDown:
LOCATE 5,1
PRINT "PROCESS: final phase - ";gencount-1;" exec functions
  defined."
PRINT#1,ges$
CLOSE 1
LOCATE 7,1
PRINT "exec.bmap now on disk. Bye-bye."
LOCATE 10,1
END
```

```
REM ****************************************
REM * intuition.bmap linker file GENERATOR *
REM ****************************************
REM *  written 6/23/86 by tobias weltner   *
REM *--------------------------------------*
REM ****************************************
REM * (C) 1986 by DATA BECKER GmbH W.-Germ.*
REM ****************************************

Header:

PRINT "[intuition.bmap] linker file GENERATOR"


FunctionDefinitions:

DATA AddGadget,9,10,1,42
DATA AllocRemember,9,1,2,396
DATA AutoRequest,9,10,11,12,1,2,3,4,348
DATA BeginRefresh,9,354
DATA BuildSysRequest,9,10,11,12,1,2,3,360
DATA ClearDMRequest,9,48
DATA ClearMenuStrip,9,54
DATA ClearPointer,9,60
DATA CloseScreen,9,66
DATA CloseWindow,9,72
DATA CloseWorkBench,78
DATA CurrentTime,9,10,84
DATA DisplayAlert,1,9,2,90
DATA DisplayBeep,9,96
DATA DoubleClick,1,2,3,4,102
DATA DrawBorder,9,10,1,2,108
DATA DrawImage,9,10,1,2,114
DATA EndRefresh,9,1,366
DATA EndRequest,9,10,120
DATA FreeRemember,9,1,408
DATA FreeSysRequest,9,372
DATA GetDefPrefs,9,1,126
DATA GetPrefs,9,1,132
```

```
DATA InitRequester,9,138
DATA IntuiTextLength,9,330
DATA ItemAddress,9,1,144
DATA MakeScreen,9,378
DATA ModifyIDCMP,9,1,150
DATA ModifyProp,9,10,11,1,2,3,4,5,156
DATA MoveScreen,9,1,2,162
DATA MoveWindow,9,1,2,168
DATA OffGadget,9,10,11,174
DATA OffMenu,9,1,180
DATA OnGadget,9,10,11,186
DATA OnMenu,9,1,192
DATA OpenScreen,9,198
DATA OpenWindow,9,204
DATA OpenWorkBench,210
DATA PrintIText,9,10,1,2,216
DATA RefreshGadgets,9,10,11,222
DATA RemakeDisplay,384
DATA RemoveGadget,9,10,228
DATA ReportMouse,9,1,234
DATA Request,9,10,240
DATA RethinkDisplay,390
DATA ScreenToBack,9,246
DATA ScreenToFront,9,252
DATA SetDMRequest,9,10,258
DATA SetMenuStrip,9,10,264
DATA SetPointer,9,10,1,2,3,4,270
DATA SetWindowTitles,9,10,11,276
DATA ShowTitle,9,1,282
DATA SizeWindow,9,1,2,288
DATA ViewAddress,294
DATA ViewPortAddress,9,300
DATA WBenchToBack,336
DATA WBenchToFront,342
DATA WindowLimits,9,1,2,3,4,318
DATA WindowToBack,9,306
DATA WindowToFront,9,312
DATA SetPrefs,9,1,2,324
DATA AllohaWorkbench,9,402
DATA end_of_functions
```

```
RESTORE FunctionDefinitions

OPEN "intuition.bmap" FOR OUTPUT AS 1

ReadLibFunc:
READ Routine$
gencount=gencount+1
LOCATE 3,1
PRINT "PROCESS: Reading         FUNCTION: ";Routine$+SPACE$(2
  0-LEN(Routine$))
IF Routine$="end_of_functions" THEN ShutDown

counter=0

ReadLoop:
READ value(counter)
IF value(counter)<20 THEN
  counter=counter+1
  GOTO ReadLoop
ELSE
  offset=value(counter)
  counter=counter-1
END IF

offset=65536&-offset
off1=INT(offset/256)
off2=offset-(256*off1)

lib$=Routine$+CHR$(0)+CHR$(off1)+CHR$(off2)

FOR loop=0 TO counter
  lib$=lib$+CHR$(value(loop))
NEXT loop

lib$=lib$+CHR$(0)

LOCATE 4,1
PRINT "PROCESS: writing         #:";gencount

ges$=ges$+lib$

GOTO ReadLibFunc

ShutDown:
LOCATE 5,1
PRINT "PROCESS: final phase - ";gencount-2;" intuition funct
  ions defined."
PRINT#1,ges$
CLOSE 1
LOCATE 7,1
PRINT "intuition.bmap now on disk. Bye-bye."
LOCATE 10,1
END
```

```
REM ****************************************
REM * dos      .bmap linker file GENERATOR *
REM ****************************************
REM *  written 6/23/86 by tobias weltner   *
REM *--------------------------------------*
REM ****************************************
REM * (C) 1986 by DATA BECKER GmbH W.-Germ.*
REM ****************************************

Header:

PRINT "[dos.bmap] linker file GENERATOR"


FunctionDefinitions:

DATA DosClose,2,36
DATA DosCreateDir,2,120
DATA DosCurrentDir,2,128
DATA DosDeleteFile,2,72
DATA DosDupLock,2,96
DATA DosExamine,2,3,102
DATA DosExNext,2,3,108
DATA DosGetPacket,2,162
DATA DosInfo,2,3,114
DATA DosInput,54
DATA DosIoErr,132
DATA DosIsInteractive,2,216
DATA DosLock,2,3,84
DATA DosOpen,2,3,30
DATA DosOutput,60
DATA DosQueuePacket,2,168
DATA DosParentDir,2,210
DATA DosRead,2,3,4,42
DATA DosRename,2,3,78
DATA DosSeek,2,3,4,66
DATA DosSetComment,2,3,180
DATA DosSetProtection,2,3,186
DATA DosUnLock,2,90
```

```
DATA DosWaitForChar,2,3,204
DATA DosWrite,2,3,4,48
DATA DosCreateProc,2,3,4,5,138
DATA DosDateStamp,2,192
DATA DosDelay,2,198
DATA DosDeviceProc,2,174
DATA DosExit,2,144
DATA DosExecute,2,3,4,222
DATA DosLoadSeg,2,150
DATA DosUnLoadSeg,2,168
DATA end_of_functions

RESTORE FunctionDefinitions

OPEN "dos.bmap" FOR OUTPUT AS 1

ReadLibFunc:
READ Routine$
gencount=gencount+1
LOCATE 3,1
PRINT "PROCESS: Reading        FUNCTION: ";Routine$+SPACE$(2
  0-LEN(Routine$))
IF Routine$="end_of_functions" THEN ShutDown

counter=0

ReadLoop:
READ value(counter)
IF value(counter)<20 THEN
  counter=counter+1
  GOTO ReadLoop
ELSE
  offset=value(counter)
  counter=counter-1
END IF

offset=65536&-offset
off1=INT(offset/256)
off2=offset-(256*off1)
```

```
lib$=Routine$+CHR$(0)+CHR$(off1)+CHR$(off2)

FOR loop=0 TO counter
  lib$=lib$+CHR$(value(loop))
NEXT loop

lib$=lib$+CHR$(0)

LOCATE 4,1
PRINT "PROCESS: writing        #:";gencount

ges$=ges$+lib$

GOTO ReadLibFunc

ShutDown:
LOCATE 5,1
PRINT "PROCESS: final phase - ";gencount-2;" dos functions d
  efined."
PRINT#1,ges$
CLOSE 1
LOCATE 7,1
PRINT "dos.bmap now on disk. Bye-bye."
LOCATE 10,1
END
```

```
REM ****************************************
REM * diskfont .bmap linker file GENERATOR *
REM ****************************************
REM *  written 6/23/86 by tobias weltner   *
REM *--------------------------------------*
REM ****************************************
REM * (C) 1986 by DATA BECKER GmbH W.-Germ.*
REM ****************************************

Header:

PRINT "[diskfont.bmap] linker file GENERATOR"


FunctionDefinitions:

DATA AvailFonts,9,1,2,36
DATA OpenDiskFont,9,30
DATA end_of_functions

RESTORE FunctionDefinitions

OPEN "diskfont.bmap" FOR OUTPUT AS 1

ReadLibFunc:
READ Routine$
gencount=gencount+1
LOCATE 3,1
PRINT "PROCESS: Reading        FUNCTION: ";Routine$+SPACE$(2
  0-LEN(Routine$))
IF Routine$="end_of_functions" THEN ShutDown

counter=0

ReadLoop:
READ value(counter)
IF value(counter)<20 THEN
  counter=counter+1
  GOTO ReadLoop

ELSE
  offset=value(counter)
  counter=counter-1
END IF
```

```
offset=65536&-offset
off1=INT(offset/256)
off2=offset-(256*off1)

lib$=Routine$+CHR$(0)+CHR$(off1)+CHR$(off2)

FOR loop=0 TO counter
  lib$=lib$+CHR$(value(loop))
NEXT loop

lib$=lib$+CHR$(0)

LOCATE 4,1
PRINT "PROCESS: writing          #:";gencount

ges$=ges$+lib$

GOTO ReadLibFunc

ShutDown:
LOCATE 5,1
PRINT "PROCESS: final phase - ";gencount-2;" diskfont functi
  ons defined."
PRINT#1,ges$
CLOSE 1
LOCATE 7,1
PRINT "diskfont.bmap now on disk. Bye-bye."
LOCATE 10,1
END
```

Wichtig: Wenn Sie in Ihrem Programm eine oder mehrere Libraries benutzen, dann müssen sich die entsprechenden ".bmap"-Files in demselben Directory befinden, da der BASIC-Befehl LIBRARY ansonsten eine "File not found"-Meldung ausgibt. Wenn Sie solche Programme auf andere Disketten kopieren, sollten Sie also immer dafür sorgen, daß auch die ".bmap"-Files kopiert werden. Unter AmigaBASIC können bis zu 5 Libraries gleichzeitig geöffnet sein.

2.3 AmigaBASIC-Grafik

Bereits im Jahre 1983 entstanden die ersten Gerüchte über einen Supercomputer der Firma Amiga. Wollte ihnen zunächst niemand Glauben schenken, so änderte sich dies schlagartig, als bekannt wurde, wer für Amigas drei Custom Chips und weite Teile der

übrigen Hardware zuständig war. Jay Miners Grafik-Chips für die Atari 400/800 Serie waren ein Begriff, als er sich bei Amiga ans Werk machte.

Amigas Grafikfähigkeiten standen also von Anfang an im Mittelpunkt und blieben von den vielen Änderungen im Konzept, die Amiga über sich ergehen lassen mußte, nahezu unberührt. Die drei Custom Chips Paula (Peripheral/Audio), Denise (Display Encoder) und Agnus (Adressgenerator) wurden tatsächlich entwickelt und bilden nun die Grundlage für Ihr Grafikwunder.

AmigaBASIC unterstützt die Amiga-Grafik fast hundertprozentig. Lediglich der fusselige Hold-And-Modify-Mode (4096 Farben gleichzeitig) sowie die neue Half-Brite-Technik (64 (fast) unabhängige Farben in low-res) sind (noch) nicht durch BASIC-Befehle programmierbar. Leider sind die Befehlsbeschreibungen zum Teil recht kurz ausgefallen, und so gibt's gleich noch etwas dazu:

2.3.1 Floodfill-Operationen

Die Befehle PAINT und AREAFILL können in Bruchteilen einer Sekunde beliebige Teile des Bildschirms ausfüllen. In verschiedenen Farben und Mustern natürlich. Die Geschwindigkeit dieser Operation, die bei anderen Computern durchaus Zeit zum Duschen lassen konnte, liegt beim Amiga an der besonderen Hardware-Architektur. Anstatt die ohnehin schwer schuftende 68000-CPU mit weiterer Arbeit zu überhäufen wird der Auftrag an einen Koprozessor abgegeben, der völlig unabhängig von der CPU seine Arbeit erledigen kann. Es handelt sich um Jay Miners "Blitter", der in "Agnus", einem der drei Amiga-Custom-Chips, vegetiert. Flächen können mit einer Million Bildpunkten pro Sekunde ausgefüllt werden. Ebenso schnell werden Linien gezogen.

2.3.2 Muster und mehr

Mit dem AmigaBASIC-Befehl PATTERN können langweilige Linien in abwechslungsreiche Punktestränge verwandelt und gähnendweiße Kreise mit ausgefallenen Mustern gefüllt werden. Wieder ist der "Blitter" zuständig, der ja eigentlich viel mehr kann, als einfach Daten kopieren.

Wie man den PATTERN-Befehl jedoch dazu bekommt, die gewünschten Muster zu produzieren, ist eine andere Frage. Im AmigaBASIC-Handbuch ist zwar ein Beispielprogramm abgebildet, eine Dokumentation fehlt allerdings. Deshalb ein bißchen Hintergrund-Information: Für das Ziehen von Linien müssen Sie 16 Beispielpunkte festlegen, die dann beim Ziehen einer Linie laufend wiederholt werden. Dasselbe gilt für gemusterte Flood-Fill-Operationen, wo Sie jedoch mehrere 16-Punkte-Beispiele übereinander stapeln können (da eine ausgefüllte Fläche 2-dimensional ist, eine Linie jedoch nur eine Dimension besitzt). PATTERN benötigt nun hexadezimale oder dezimale Werte. Die folgende binäre Information für eine gepunktete Linie:

* * * * * * * *

müßte also zunächst umgerechnet werden:

2^15+2^11+2^9+2^7+2^5+2^3+2^1

= 43690

Noch schwieriger wird es, wenn man versucht, diese Werte in einem Integer-Array unterzubringen, da dort alle Zahlen größer (2^15)-1 = 32767 negativ dargestellt werden müssen. Es handelt sich also um einen Zahlenkreis gegenüber dem üblichen Zahlenstrang.

Die folgende Unterroutine, die Sie einfach in Ihre eigene Programme übernehmen können, beschert Ihnen die Funktion Set-Pattern, die alle Rechenarbeit für Sie übernimmt. Das Aufruf-Format lautet:

```
SetPattern nummer,"xxxxxxxxxxxxxxxx"
```

nummer: Ist diese Nummer negativ, so wird das Aussehen von Linien verändert. Ist sie 0 oder positiv, dann handelt es sich um die Ebene der Fill-Definition.

string: Der nachfolgende String enthält den Status der 16 Bits, also das Aussehen Ihrer Definition. Ein "*" bedeutet einen gesetzten Punkt, alle anderen Zeichen werden als gelöschter Punkt interpretiert. Obwohl die SUB-Routine dies notfalls ausgleichen kann, sollte der String immer 16 Zeichen enthalten.

Programm-Größe: 1044 Bytes

```
'SetPattern

var:

DIM SHARED AREA.PAT%(7) '2^8 Linien

patDef:

SetPattern  0,"** *************"
SetPattern  1,"** *************"
SetPattern  2,"** *************"
SetPattern  3,"                "
SetPattern  4,"********* ******"
SetPattern  5,"********* ******"
SetPattern  6,"********* ******"
SetPattern  7,"                "
SetPattern -1,"* ***** * ***** "

PATTERN LinePattern%, AREA.PAT%
```

```
loop:

COLOR 2,0 'Zeichenfarbe schwarz
r=INT(RND(1)*40)+30 '30...70
x=INT(RND(1)*550)+50 '50...600
y=INT(RND(1)*130)+50 '50...180
CIRCLE (x,y),r
PAINT (x,y),3,2
COLOR 1,0 'Zeichenfarbe weiss
CIRCLE (x,y),r
GOTO loop

'-------------------------------------

SUB SetPattern(nummer%,pat$) STATIC

SHARED LinePattern%

length=LEN(pat$)
result=0

FOR loop%=1 TO length
  check$=MID$(pat$,loop%,1)
  IF check$="*" THEN
    result=result+2^(16-loop%)
  END IF
NEXT loop%
IF result>32767 THEN
  result=result-2^16
END IF
IF nummer%<0 THEN
  LinePattern%=result
ELSE
  AREA.PAT%(nummer%)=result
END IF

END SUB
```

Mögliche Fehlerquellen:

TYPE MISMATCH bei einem Aufruf der SetPattern-Funktion:

SetPattern erwartet eine Integer-Variable (-32768 bis +32767), ein Komma und einen String. Falls dies ordnungsgemäß erfolgte, so ist der Grund eine falsche Variablen-Angabe in der SUB-Deklaration. Fehlt das %-Zeichen hinter "nummer"?

SUBSCRIPT OUT OF RANGE innerhalb der SUB-Routine:

Es wurden zu Anfang des Programms durch DIM SHARED weniger Elemente reserviert als durch SetPattern definiert. SetPattern 13,"****************" bei nur 7 dimensionierten Elementen funktioniert beispielsweise nicht. Erhöhen Sie die Elemente in der DIMensionierung gemäß der Regel (2^x)-1.

ILLEGAL FUNCTION CALL bei Benutzung des PATTERN-Befehls:

Hier ist die Lage eindeutig. Sie haben weniger als 2^x Ebenen definiert. Die DIM SHARED Anweisung zu Anfang des Programms muß (2^x)-1 Elemente festlegen (0,1,3,7,15,31,63,etc.). Dies wurde hier unterlassen.

Variablen-Felder:

AREA.PAT%: Dieses Feld muß vor Gebrauch definiert werden. Es enthält die Definitionsebenen für Fill und muß immer 2^x Ebenen beinhalten. Es gilt: DIM SHARED AREA.PAT%((2^x)-1). Zwei Beispiele verdeutlichen das:

DIM SHARED AREA.PAT%(3) ist zulässig (3+1=(2^2)) definiert 4 Ebenen.

DIM SHARED AREA.PAT%(5) ist unzulässig (5+1 kein Vielfaches von 2).

Variablen:

LinePattern%:	Enthält die Liniendefinition
x,y,r	bezeichnen x- und y-Koordinate des Demo-Kreises sowie den Radius
nummer%	ist die SUB-Variable für die Nummer der Ebene, die definiert werden soll
pat$:	Weitere SUB-Variable, enthält binäre Darstellung dieser Ebene
length:	SUB-Variable, Länge der pat$-Variable
result:	SUB-Variable, Dezimalwert binäre Maske in pat$

Ein Integer-Variablefeld AREA.PAT% wird angelegt und als SHARED dimensioniert. So stehen die Variablen auch SUB-Programmen zur Verfügung. Das Feld enthält 8 Elemente (2^3) und soll später die Definitionsebenen für die Fill-Operationen aufnehmen.

Es folgen die Definitionen, die den neuen SUB-Befehl SetPattern benutzen. Zuerst werden die acht Ebenen (0 - 7) für die Fill-Operationen mit Backstein-Mustern belegt, danach werden alle Linien (-1) mit einem wundervollen Punktemuster versehen. Es folgt der eigentliche PATTERN-Befehl, der die von SetPattern berechneten Variablen LinePattern% und AREA.PAT% einsetzt.

Damit die Sache auch einen Sinn hat, folgt ab loop: ein kleines Beispielprogramm, das nach dem Zufallsprinzip Kreise auf den Bildschirm malt, die dann mit unserem geistvollen Backstein-Muster gefüllt werden. Mit dem PAINT-Befehl wird dabei zunächst ein schwarzer Kreis gezeichnet, den das Programm als Füllgebiet ansieht. Dann wird der Kreis wieder weiß eingefärbt, damit spätere schwarze Füllgebiete nicht behindert werden. Nur

so ist es von BASIC aus möglich, Strukturen ohne Rücksicht auf den Hintergrund vollständig auszufüllen. Falls Sie der schwarze Kreis stört, geben Sie bitte zu Anfang des Programms

```
PALETTE 2,1,1,1
```

ein. Dadurch wird die Farbe Schwarz auch weiß und fällt im Bild gar nicht mehr auf.

2.3.3 Zeichenmodi verändern

Ob Sie es bemerkt haben oder nicht: Der Amiga kennt vier verschiedene Zeichenmodi. Wenn Sie Grafik auf dem Bildschirm erzeugen, dann kann diese vom Rechner auf vier grundsätzliche Weisen interpretiert werden:

JAM 1: Wenn Sie eine Grafik zeichnen (und dazu gehört auch die Ausführung eines einfachen PRINT-Befehls), dann wird nur eine Farbe, die Zeichenfarbe, ins Zielgebiet gemalt. Für jeden gezeichneten Punkt wird die Farbe an dieser Stelle verändert und alle anderen Punkte bleiben unberührt (nur eine Farbe wird ins Zielgebiet "gejammed").

JAM 2: Hier werden zwei Farben ins Zielgebiet gemalt. Ein gesetzter Punkt wird in der Vordergrundfarbe (AmigaBASIC-Farbregister 1) gemalt, ein ungesetzter Punkt nimmt die Farbe des Hintergrundes an (AmigaBASIC-Farbregister 0). Der Grafikhintergrund wird also durch Ihre Aktionen beeinflußt, zwei Farben werden "gejammed".

COMPLEMENT: Dieser Mode verfährt genau wie JAM1, jedoch wird der Punkt nicht mit AmigaBASIC-Farbregister 1 ausgefüllt, sondern invertiert (complemented). Ein ursprünglich gesetzter Punkt wird gelöscht, und umgekehrt.

INVERSEVID: AmigaBASIC-Farbregister 0 und Farbregister 1 werden einfach vertauscht. Die Folge ist das altbewährte Invertieren des Bildschirms.

Zusätzlich können alle vier Modi miteinander gemischt werden, also in JAM1|COMPLEMENT oder JAM2|COMPLEMENT|INVEREVID resultieren. Es stehen neun verschiedene Kombinationsmöglichkeiten zur Verfügung. Über den Sinn so mancher Kombinationen läßt sich bekanntlich streiten...

AmigaBASIC besitzt zur Zeit keinen Befehl, um willkürlich den Zeichenmode zu verändern. Durch den LIBRARY-Befehl läßt sich dieser außerordentlich wichtige Befehl nachträglich implementieren. Einzige Voraussetzung: Sie besitzen ein "graphics.bmap"-File zusammen mit AmigaBASIC auf Disk (näheres dazu beim bmap-Programmlisting am Anfang dieses Kapitels).

Der in der "graphics.library" definierte Befehl SetDrMd (Set Draw Mode) läßt sich für unsere Zwecke ganz vorzüglich mißbrauchen. Er hat das Format:

```
SetDrMd (RastPort,Mode)
```

Die Variable RastPort muß einen Zeiger auf die RastPort-Struktur unseres Windows enthalten. Dieser Zeiger ist immer in WINDOW(8) gespeichert. Das AmigaBASIC-Format sieht also so aus:

```
SetDrMd (WINDOW(8),Mode)
```

Mode:

```
JAM1       : 0
JAM2       : 2^0=1
COMPLEMENT : 2^1=2
INVERSEVID : 2^2=4
```

Um Modes zu mischen, addiert man einfach die entsprechenden Zahlen. Mode=3 entspricht z.B. JAM2|COMPLEMENT. Hier wird auch deutlich, wie verhindert wird, die absolut sinnlose Kombination JAM1|JAM2 zu selektieren: 0+1 bleibt 1.

```
'SetDrawMode

LIBRARY "graphics.library"

main:

FOR anzahl%=1 TO 3
  LOCATE 1,1
  FOR test%=0 TO 9
    SetDrawMode test%
    PRINT TAB(anzahl%)"HALLO!"
    LOCATE test%+1,30
    PRINT "Draw-Mode: ";test%
  NEXT test%
NEXT anzahl%

LIBRARY CLOSE
END

'---------------------------------

SUB SetDrawMode (mode%) STATIC
  CALL SetDrMd&(WINDOW(8),mode%)
END SUB
```

2.3.4 Shadow-Print

Vor einiger Zeit haben sich Fachleute Gedanken darüber gemacht, wie man die Darstellungsqualität von Texten verbessern kann, die zum Beispiel bei der Sportreportage ins Fernsehbild geblendet werden. Man ist auf eine ebenso einfache wie erfolgreiche Methode gekommen: Der Text wird einfach schattiert, wodurch der Kontrast natürlich erheblich verbessert wird. So kann man, völlig unabhängig vom Hintergrund, gut lesbare Schrift produzieren.

Was das Fernsehen kann, kann unser Amiga schon lange. Das folgende Programm implementiert den Shadow-Befehl, der sich nicht nur für Genlock hervorragend eignet:

```
Shadow a$,spacing
Shadow "hello",spacing
```

"spacing:" enthält Abstand der Buchstaben

Programmgröße: 673 Bytes

Bemerkungen: "graphics.bmap" muß sich auf Disk befinden

```
'Shadow-Print

DECLARE FUNCTION Move& LIBRARY
LIBRARY "graphics.library"

main:

Shadow "HALLO!",10
Shadow "Dies ist ein Test.",7
Shadow "Extended Spacing!",14
PRINT

LIBRARY CLOSE
END

'------------------------------------

SUB Shadow(text$,spacing%) STATIC

depth%=1
crsX%=POS(0)
crsY%=(CSRLIN)*8               'Cursor-Koordinaten
IF crsY%<8 THEN crsY%=8        'erste BS-Zeile ins Format
```

```
CALL SetDrMd&(WINDOW(8),0)    'JAM1
FOR loop%=1 TO LEN(text$)
  b$=MID$(text$,loop%,1)
  e&=Move&(WINDOW(8),crsX%+depth%,crsY%+depth%)
  COLOR 2,0                   'Schatten (schwarz) malen
  PRINT b$;
  COLOR 1,0
  e&=Move&(WINDOW(8),crsX%,crsY%)
  PRINT b$;                   'Zeichen malen
  crsX%=crsX%+spacing%
NEXT loop%
PRINT
CALL SetDrMd&(WINDOW(8),1)    'JAM2 (normal)

END SUB
```

Mögliche Fehlerquellen:

FILE NOT FOUND bei Aufruf der LIBRARY-Funktion:

Das nötige File "graphics.bmap" befand sich nicht im gleichen Directory mit AmigaBASIC und konnte daher vom Programm nicht gefunden werden. Benutzen Sie CHDIR oder kopieren Sie das bmap-File ins nötige Directory.

OVERFLOW bei Aufruf der Move-Routine:

Move wurde als Move& deklariert. Sie haben das &-Zeichen oder die erste Zeile des Programms vergessen.

SOFTWARE FAILURE:

Sie haben einen der Library-Befehle mit falschen Argumenten aufgerufen (insbesondere evtl. WINDOW(8) als Rastport vergessen) oder Ihr "graphics.bmap"-File enthält einen Fehler.

TYPE MISMATCH bei Aufruf des Shadow-Befehles:

In der SUB-Deklaration fehlt das %-Zeichen als Deklaration der Variablen spacing%, oder Sie haben versucht, Fließkomma-Variablen (1.3, 7.23,...) als spacing zu benutzen.

Variablen:

text$:	SUB-Variable, enthält den gewünschten Text
spacing%:	SUB-Variable, enthält den Pixelabstand der einzelnen Buchstaben.
depth%:	SUB-Variable, enthält die "Tiefe" des Schattens
x%:	SUB-Variable, x-Position des AmigaBASIC-Cursors
y%:	SUB-Variable, y-Position des AmigaBASIC-Cursors
b$:	SUB-Variable, Ausgefiltertes Zeichen, das gedruckt werden soll
e&:	SUB-Variable, ERROR-Message des Move&-Befehls

Unser Programm benötigt die Library-Funktionen SetDrMd, die wir gerade abgehandelt haben, sowie Move. Da Move einen Wert an AmigaBASIC zurückliefert, muß dieser Befehl zunächst als Funktion deklariert werden. Anschließend wird die "graphics.library" für BASIC eröffnet. Zwei Beispieltexte werden ausgegeben, die die verschiedenen Zeichenabstände demonstrieren. Anschließend wird die Library wieder geschlossen und das Programm endet.

Die SUB-Routine Shadow übernimmt zwei Variablen: Den Text und den Zeichenabstand, der im Normalfall 8 betragen sollte (Font topaz.8). Die Tiefe des Schattens wird festgelegt (experimentieren Sie gelegentlich einmal mit diesem Wert herum), die aktuellen Cursorpositionen werden festgelegt, und der Zeichenmode auf JAM1 gestellt (sonst wäre es ja nicht möglich, den Schatten mit weißer Farbe zu überschreiben, ohne ihn gänzlich auszulöschen). Die folgende Schleife gibt Zeichen für Zeichen aus, wobei das schwarze Schattenzeichen um depth% nach rechts und unten verschoben ist. Anschließend wird der Zeichenmode wieder auf JAM2, den Normallfall, gestellt.

2.3.5 Fettdruck

Eine weitverbreitete Möglichkeit, Texte hervorzuheben, ist die des Fettdrucks. AmigaBASIC besitzt diese eigensinnige Darstellungsart zwar von Natur aus nicht, aber dank des SetDrMd-Befehls kann auch diese rasch implementiert werden. Unser neuer Befehl heißt:

```
Bold a$,spacing
Bold "Hello World!",spacing
```

wobei spacing wieder den Buchstaben-Abstand festlegt. Da diese Funktion in wesentlichen Punkten dem Shadow-Befehl ähnelt, schlagen Sie bitte für eine ausführliche Dokumentation der angewandten Techniken dort nach.

Programm-Größe: 572 Bytes
Bemerkungen: "graphics.bmap" muß sich auf Disk befinden

```
DECLARE FUNCTION Move& LIBRARY

LIBRARY "graphics.library"

main:

FOR test%=1 TO 8
  LOCATE test%,5
  Bold "Hello Mrs. Wilke!",test%+6
NEXT test%

LIBRARY CLOSE
END
```

```
SUB Bold(text$,spacing%) STATIC

x%=POS(0)
y%=(CSRLIN)*8
CALL SetDrMd(WINDOW(8),0)
FOR loop=1 TO LEN(text$)
  b$=MID$(text$,loop,1)
  e&=Move&(WINDOW(8),x%,y%)
  PRINT b$;
  e&=Move&(WINDOW(8),x%+1,y%)
  PRINT b$;
  x%=x%+spacing%
NEXT loop
CALL SetDrMd(WINDOW(8),1)

END SUB
```

Allgemeine Programm-Informationen: Siehe "Shadow-Print".

Der Fettdruck-Effekt beruht auf der folgenden Technik: Der Text wird normal ausgeprintet. Anschließend wird derselbe Text um einen Bildschirmpunkt nach rechts verschoben nocheinmal gedruckt. Dieser "smear"-Effekt hebt den Text hervor.

2.3.6 Outline-Druck

Die Bold-Funktion hebt den Text auf recht einfache Weise hervor. Diese Funktion, die wir "Outline" genannt haben, macht dasselbe mit ein wenig mehr Stiel. Lassen Sie sich überraschen! Die Funktion wird durch:

```
Outline a$,spacing
Outline "Hello Mrs. Wilke!",spacing
```

aufgerufen.

Programm-Größe: 710 Bytes
<u>Bemerkungen:</u> "graphics.bmap" muß sich auf der Disk befinden.

```
DECLARE FUNCTION Move& LIBRARY

LIBRARY "graphics.library"

main:

FOR test%=1 TO 8
  LOCATE test%,5
  Outline "Cheer up, ol´ Begger!",test%+6
  LOCATE test%,40
  Outline "Size: "+STR$(test%+6),8
NEXT test%


LIBRARY CLOSE
END

´-------------------------------------

SUB Outline(text$,spacing%) STATIC

mode%=2 ´versuchen Sie auch mal 5 und 6!
x%=POS(0)*8
y%=(CSRLIN)*8
FOR loop=1 TO LEN(text$)
  b$=MID$(text$,loop,1)
  CALL SetDrMd(WINDOW(8),0)
  FOR yoffset%=-1 TO 1
    FOR xoffset%=-1 TO 1
      e&=Move&(WINDOW(8),x%+xoffset%,y%+yoffset%)
      PRINT b$;
    NEXT xoffset%
  NEXT yoffset%
  CALL SetDrMd(WINDOW(8),mode%) ´invertieren
  e&=Move&(WINDOW(8),x%,y%)
  PRINT b$;
  x%=x%+spacing%
NEXT loop
CALL SetDrMd(WINDOW(8),1)

END SUB
```

Allgemeine Programm-Informationen: Siehe "Shadow-Print"

Der Outline-Effekt ist zwar ein bißchen rechenintensiv, aber er macht sich wirklich bezahlt. Das Programm entspricht in wesentlichen Punkten der Bold-Funktion: Der Text wird in alle vier Himmelsrichtungen verschmiert. Anschließend kommt unser glorreicher Zeichenmodus COMPLEMENT zum Zuge, der das Aussehen des wirklichen Buchstabens in die verschmierte Maske komplementiert. Es ist wirklich schwer zu erklären, schauen Sie sich am besten das Programm mit der Einzelschritt-Funktion (linker AMIGA-Key und T) an.

Übrigens: Als kleinen Leckerbissen empfehlen wir Ihnen, den Zeichenmode COMPLEMENT in der Variable mode% mit anderen Zeichenmodi zu vermischen. Insbesondere die Werte 5 und 6 (COMPLEMENT|JAM2 bzw. COMPLEMENT|INVERSEVID) verändern den Outline-Effekt, ohne ihn unlesbar zu machen.

2.3.7 Andere Druck-Modi

Mit dem SetDrMd-Befehl läßt sich noch eine ganze Menge mehr machen. Da die Programmstruktur weitgehend den vorangegangenen "Befehls-Erweiterungen" entspricht, haben wir darauf verzichtet, einen Befehl wie Reverse (SetDrMd mode% = INVERSEVID) oder Underline (SetDrMd mode%=JAM1, dann jeden Buchstaben ausgeben, anschließend darüber das Unterstreichungssymbol) zu implementieren. Vielleicht haben Sie ja Lust, mit diesen Anregungen weiterzuarbeiten.

2.3.8 Move-Kontrolle über den AmigaBASIC-Cursor

In einigen der vorangegangenen Befehle haben wir ihn schon benutzt, den "graphics.library"-Befehl MOVE. Kennt AmigaBASIC nur die Möglichkeit, die Cursorposition zeichenweise (LOCATE-Befehl) oder pixelweise in x-Richtung (durch PTAB)

zu verändern, so läßt sich die Cursorposition mit Hilfe des Move-Befehls problemlos pixelweise sowohl in x- als auch y-Richtung verschieben.

Der Aufruf des Befehls in BASIC muß folgendermaßen erfolgen:

```
Move&(WINDOW(8),x%,y%)
```

Um die Sache zu vereinfachen, haben wir wieder einen kleinen Befehl geschrieben, der in den verschiedensten Situationen äußerst praktisch sein kann:

```
xyPTAB x%,y%
```

Programm-Größe: 455 Bytes
Bemerkungen: "graphics.bmap" muß sich auf Disk befinden.

```
DECLARE FUNCTION Move& LIBRARY

LIBRARY "graphics.library"

var:
text$="Es geht bergab..."
text$=" "+text$+" "
empty$=SPACE$(LEN(text$))
fontheight%=8

main:
FOR y%=6 TO 100
  xyPTAB x%,y%
  PRINT text$
  xyPTAB x%,y%-fontheight%
  PRINT empty$
  x%=x%+1
NEXT y%

LIBRARY CLOSE
END
```

```
SUB xyPTAB(x%,y%) STATIC
  e&=Move&(WINDOW(8),x%,y%)
END SUB
```

Fehlerquellen: Schwer möglich.

Variablen:

text$:	Demo-Text
empty$:	Leerstring, der für ein restloses Verschwinden beim y-Verschieben sorgt
fontheight%:	Höhe des Font
x%, y%:	Bildschirm-Koordinaten
e&:	error-Meldung des Move&-Befehls

Programmbeschreibung:

Der Move&-Befehl wird als Funktion deklariert, und die einschlägige Library wird geöffnet. Der Demo-Text huscht im SoftScroll-Mode über den Schirm, die Library wird wieder dichtgemacht, und das Programm endet.

Die eigentliche SUB-Routine ist denkbar einfach, denn im Prinzip werden nur dem Move-Befehl die nötigen Koordinaten übergeben.

So simpel diese Routine aussieht, so leistungsfähig ist sie. Mit ihrer Hilfe kann Text, wie im Beispiel, pixelweise in sämtliche Himmelsrichtungen verschoben werden. Entweder im Smear-Effekt (SetDrMd mode%=JAM1) oder als SoftScrolling (SetDrMd mode%=JAM2).

2.3.9 Rubberband-Effekt

Was, Sie wissen nicht, was das ist? Dabei arbeiten Sie damit sicherlich täglich: Jedesmal, wenn Sie die Größe eines Windows verändern, erscheint dieses erfrischend orange Gummiband mit dessen Hilfe Sie eine passende Größe finden können.

Dieses Gummiband wird normalerweise von Intuition verwaltet. Die Technik ist recht einfach: Um zu verhindern, daß durch das Gummiband der Bildhintergrund verändert wird, friert Intuition zunächst alle Bildaktivitäten ein (Dies ist der Grund, warum ein Malprogramm zum Beispiel die Arbeit unterbricht, wenn Sie ein Window vergrößern oder -verkleinern). Das Gummiband wird anschließend im Zeichen-Mode COMPLEMENT auf den Bildschirm gezeichnet. So kann es durch einfaches Überschreiben problemlos wieder gelöscht werden, ohne den Bildhintergrund zu verändern.

Dieser Effekt läßt sich auch von BASIC aus recht einfach programmieren. Das folgende Programm ermöglicht einen Einblick und benutzt auch gleich noch ein paar andere interessante AmigaBASIC-Befehle:

Programm-Größe: 1503 Bytes
Bemerkungen: "graphics.bmap" muß sich auf Disk befinden.

```
'rubberband

LIBRARY "graphics.library"

var:
DIM picture%(10000)
PALETTE 0,0,0,0

init:

GOSUB timeCheck
ON MOUSE GOSUB MouseCheck
MOUSE ON
ON TIMER(.1) GOSUB timeCheck
TIMER ON
```

```
circlel:

radius=INT(RND(1)*100)+10
FOR x=0 TO 3.1415926# STEP .05
  col=INT(RND(1)*3)+1
  x1=radius*COS(x)
  y1=radius*SIN(x)
  x2=radius*COS(x+3.1415926#)
  y2=radius*SIN(x+3.1415926#)
  LINE ((150+x1)*2,100+y1)-((150+x2)*2,100+y2),col
NEXT x
GOTO circlel

MouseCheck:

CALL SetDrMd(WINDOW(8),2)

loop:

mstat=MOUSE(0)
IF mstat<1 THEN
  mox=MOUSE(1)
  moy=MOUSE(2)
  mx=mox

  my=moy
END IF

checkloop:
IF mstat<1 THEN
  mtx=mx
  mty=my
  mx=MOUSE(1)
  my=MOUSE(2)
  IF 6+((my-moy+1)*2*INT((mx-mox+16)/16))>ABS(7000) THEN
    mx=mtx
    my=mty
  END IF
  IF mtx<>mx OR mty<>my THEN
    LINE (mox,moy)-(mtx,mty),,b
    LINE (mox,moy)-(mx,my),,b
  END IF
  mstat=MOUSE(0)
  GOTO checkloop
END IF

LINE (mox,moy)-(mx,my),,b
PSET (mox,moy)
```

```
CALL SetDrMd(WINDOW(8),1)

GET (mox,moy)-(mx,my),picture%
PUT (0,0),picture%,PSET

RETURN

timeCheck:

IF fl=1 THEN
  c=c-.0625
  IF c<.0625 THEN fl=0
ELSEIF fl=0 THEN
  c=c+.0625
  IF c>.9375 THEN fl=1
END IF
PALETTE 1,.8,.8,c    'Farbtabelle fuer
PALETTE 2,c,.9,c     'Rotation
PALETTE 3,.6,c,.7

RETURN
```

Variablen-Felder:

picture%: Enthält das selektierte Bildschirm-Image.

Variablen:

radius:	Zufallswert für Kreis-Radius
col:	Zufallswert für Zeichenfarbe
x1,y1:	x- und y-Koordinaten für Linienbeginn
x2, y2:	x- und y-Koordinaten für Linienende
mstat:	Status des linken Maus-Buttons
mox, moy:	Koordinaten des Maus-Ursprungs
mtx, mty:	Temporäre Maus-Koordinaten
mx, my:	Aktuelle Maus-Koordinaten
c:	Farbschleife im Timer-Interrupt

Programmbeschreibung:

Dieses Programm soll den Rubberband-Effekt demonstrieren. Dazu ist der SetDrMd-Befehl aus der "graphics.library" erforderlich. Das Feld picture% wird dimensioniert und die Hintergrundfarbe gesetzt. Der Maus-Interrupt wird eingeschaltet. Außerdem wird ein Timer-Interrupt eingerichtet, der alle 1/10 Sekunden die Routine timeCheck ausführt.

Der Programmteil demo: zeichnet nun verschiedenfarbige Linienkreise auf den Bildschirm. Die beiden Funktionen COS und SIN liefern dabei die nötigen Kreiskoordinaten, die im nachfolgenden LINE-Befehl Verwendung finden (die x-Werte werden außerdem um den Faktor 2 vergrößert, da wir kein Ei, sondern einen wundervoll symmetrischen Kreis im Sinn haben. AmigaBASICs CIRCLE-Befehl gleicht die Auflösungsunterschiede automatisch aus).

MouseCheck enthält die eigentliche Rubberband-Routine: Zunächst wird der Zeichenmode auf COMPLEMENT (=2) geschaltet, damit das Programm das Gummiband später auch gefahrenlos wieder aus dem Bildschirm herausbekommt. Anschließend werden der Maus-Status getestet und die Ursprungskoordinaten auf die derzeitige Maus-Position gesetzt. Danach beginnt die Routine eine Schleife, die zunächst die alte Mausposition zwischenspeichert und dann die neue holt. Anschließend

wird die umschlossene Fläche berechnet, damit sie auch in unser Feld picture% hineinpaßt. Ist die Fläche zu groß, dann werden einfach die alten Koordinaten beibehalten.

Nun werden die aktuellen und temporären Mauspositionen miteinander verglichen, um festzustellen, ob sich die Maus seit dem letzten Check vom Fleck gerührt hat. Falls ja, so wird das alte Gummiband gelöscht und ein neues mit den aktuellen Koordinaten gezeichnet. Wieder wird der Maus-Status kontrolliert, und die Schleife wiederholt sich. Das geht solange, bis es dem Benutzer zu dumm wird und er den linken Maus-Button losläßt. Dann nämlich wird die Variable "mstat" größer als 0 und die Schleife wird verlassen.

Der Benutzer hat also das Feld markiert, und das letzte Gummiband kann gelöscht werden. Der Zeichenmode wird wieder auf JAM2 (=1) zurückgeschaltet, das ehemals umrandete Gebiet wird mittels GET ausgelesen und wandert ins Variablen-Feld picture%. PUT wirft es wieder aus, und zwar in der linken oberen Ecke des Bildschirmes (0,0). Anschließend wird die Maus-Interrupt-Routine verlassen, und das Demoprogramm setzt seine Arbeit fort.

Die Routine timeCheck ist noch eine Zugabe am Rande, die den außerordentlich praktischen Zeit-Interrupt vorstellt. Jede 1/10 sec. werden hier nämlich die Farbregister verändert.

2.4 Die Amiga-Zeichensätze

Vielfalt macht Programme erst schön. Wem kann ein einheitlicher Schriftsatz nicht schon einmal auf die Dauer auf den Keks gehen? Immer die gleichen Buchstaben auf dem Monitor zu sehen, daß brennt sich doch ein in die müden Augen des sonst allem trotzenden Programmierers. Das haben sich wahrscheinlich auch die Entwickler des Amigas gedacht, und so wurde ein Ausweg gleich mit eingebaut. Es handelt sich um sogenannte "Fonts".

Haben andere Computer oftmals nur einen einzigen, fest eingebauten Zeichensatz, so hat der Amiga gleich zwei. Und war es bei anderen Computern, wenn überhaupt, nur durch außerordentlich geschickte Bit- und Chip-Manipulationen möglich, einen weiteren Zeichensatz nachzuladen, so verwendet der Amiga disk-residente Zusatz-Fonts, die problemlos und ganz nach belieben nachgeladen werden können. Ohne Rumpokerei, ohne Grübeln über freie Speicherbereiche. Die beiden eingebauten Zeichensätze bestechen nicht gerade durch ihr künstlerische Raffinesse, sondern vielmehr durch die durchdachte Funktionalität. Wie Sie sicher wissen, ist es möglich, den Amiga-Display-Mode zwischen 60 und 80 Zeichen pro Zeile hin- und herzuschalten. Sieht man sich die Sache näher an, so wird einfach zwischen den beiden eingebauten Zeichensätzen umgeschaltet. Genial, nicht? Nein? Naja, auf diese Weise kommt der BASIC-Programmierer immerhin schon einmal in den Genuß zweier unterschiedlicher Zeichensätze:

2.4.1 60- oder 80 Zeichen, ist die Frage

Um einen der ROM- (oder sollten wir sagen: Write Once Memory?) Zeichensätze zu bekommen, spielt die "graphics.library" mal wieder Retter in der Not (da AmigaBASIC für solch geschickte Operationen natürlich keinen Befehl erfunden hat.). Wir brauchen die folgenden Funktionen:

OpenFont
CloseFont
SetFont

Wieder haben wir für Sie einen SUB-Befehl entwickelt, mit dem Sie nach herzenslust zwischen den beiden Zeichensätzen herumschalten können. Die Funktion wird durch:

```
RomFont x
```

aufgerufen.

x=60: 60 Zeichen/Zeile-Font
x=80: 80 Zeichen/Zeile-Font

Programm-Größe: 626 Bytes
Bemerkungen: "graphics.bmap" muß sich auf Disk befinden.

```
DECLARE FUNCTION AskSoftStyle& LIBRARY
DECLARE FUNCTION OpenFont& LIBRARY
DECLARE FUNCTION SetSoftStyle& LIBRARY

LIBRARY "graphics.library"

test:
ROMFont 60
PRINT "Hallo, dies ist der erste eingebaute Zeichensatz"
PRINT "mit 60 Zeichen pro Zeile mit 9*8 Matrix."
ROMFont 80
PRINT "Der zweite ist kleiner und bringt ganze 80 Zeichen"
PRINT "in einer Zeile unter. Natuerlich lassen sich die"
PRINT "beiden Zeichensaetze auch ";
ROMFont 60
PRINT "untereinander mischen!"

LIBRARY CLOSE
END


SUB ROMFont(which%) STATIC

SHARED strFont&

IF which%=60 THEN
  height%=9
ELSE
  height%=8
END IF
```

```
IF strFont&<>0 THEN CALL CloseFont(strFont&)
font0$="topaz.font"+CHR$(0)
textAttr&(0)=SADD(font0$)
textAttr&(1)=height%*2^16
strFont&=OpenFont&(VARPTR(textAttr&(0)))
IF strFont&<>0 THEN CALL SetFont (WINDOW(8),strFont&)

END SUB
```

Fehlerquellen:

FILE NOT FOUND bei Aufruf der LIBRARY Funktion:

Das nötige File "graphics.bmap" befand sich nicht im gleichen Directory mit AmigaBASIC und konnte daher vom Programm nicht aufgespürt werden. Benutzen Sie doch einfach CHDIR, oder schauen Sie einmal selber nach, ob sich das File überhaupt auf Disk befindet (Software-Klau wird ja auch immer schlimmer...).

OVERFLOW bei Aufruf eines unserer neuen Befehle

Amiga interpretiert den Befehl aus einem unerfindlichen Grund als Variable, die als solche zugegebenermaßen etwas groß geraten wäre. Überprüfen Sie, ob der Name des Befehls mit dem in der Deklaration übereinstimmt (falls es sich um eine Funktion handelt), oder entfernen Sie &- oder ähnliche Zeichen (wenn es sich um keine Funktion handelt).

SOFTWARE FAILURE:

Das ist immer ärgerlich. Entweder wurde einer unserer hilfreichen neuen Befehle falsch aufgerufen, oder "graphics.bmap" ist fehlerhaft.

TYPE MISMATCH bei Aufruf der SUB-Routine:

In der SUB-Deklaration fehlt das %-Zeichen der which%-Variablen, oder sollten Sie etwas wie

```
ROMFont 80.345
```

eingegeben haben? Soetwas sollte natürlich nicht vorkommen.

Variablen-Felder:

textAttr&(): SUB-Feld, in C wäre dieses Feld als textAttr-Struktur definiert. Hier werden alle wichtigen Informationen des Fonts untergebracht.

Variablen:

which%: SUB-Variable, enthält ID des ersehnten Font
strFont&: Zeiger zur Verwaltungsstruktur für Fonts, sollte von Ihnen nicht angerührt werden, da der Amiga sonst ins Schleudern kommt.
height%: Die Pixelhöhe des gewünschten Font. Hier gibt's nur 8 oder 9, da der Amiga Phantasie-Zahlen mit dem nächstbesten Font abspeist.

Programmbeschreibung:

Da der Befehl OpenFont& einen Wert an AmigaBASIC zurückliefert, muß er zunächst als Funktion deklariert werden. Dann wird die "graphics.library" geöffnet, die uns all diese wundervollen Dinge ermöglicht. Ein kleines Demo-Programm demonstriert den prinzipiellen Unterschied beider Zeichensätze. Die Library wird sinnvollerweise wieder geschlossen, und das Programm endet.

Der SUB-Befehl verlangt die Kennzeichnung des gewünschten Fonts: 60 oder 80. Diese wird in which% gespeichert. Unser Amiga kümmert sich natürlich nicht um Angaben wie "Zeichen/Zeile", sondern will die Höhe des Fonts wissen. Diese be-

trägt 8 bzw. 9 Bildschirmpunkte. Etwaige offene Zeichensätze werden geschlossen, und es wird eine textAttr-Struktur initialisiert. Eine solche Struktur ist in C als:

```
STRPTR    ta Name;
UWORD     ta YSize;
UBYTE     ta Style;
UBYTE     ta Flags;
```

definiert. Als Speicherplatz verwenden wir von BASIC einfach das Variablen-Feld textAttr&(). Die ersten vier Bytes enthalten, gemäß der Struktur, einen Zeiger auf den Namen des Fonts, der mit 0 abgeschlossen sein muß: font0$. Die nächsten zwei Bytes werden mit der Höhe des Fonts belegt (deshalb muß height% mit 2^16 multipliziert werden).

Nachdem diese lebenswichtige Struktur mit unseren Parametern versorgt ist, kann getrost versucht werden, den Font zu öffnen. Im Erfolgsfall liefert dabei OpenFont& einen Zeiger zum neuen Verwalter, ansonsten kommt 0 zurück. Falls alles geklappt hat, wird der neue Font eingeschaltet, was mit SetFont geschieht.

2.4.2 Algorithmisch gesteuerte Zeichensätze

Es klingt schlimmer als es ist. Neben der Fähigkeit, völlig neue Zeichensätze auf den Bildschirm zu zaubern, kann der Amiga die bestehenden Fonts algorithmisch, also auf mathematische Weise, verändern. Dabei stehen die folgenden Manipulationen zur freien Verfügung, die natürlich auch miteinander gemischt werden können:

0. Normale Darstellung (auch ganz nützlich...)
1. Unterstrichen (underline)
2. Fettdruck (bold)
3. Schrägschrift (italics)

Durch die Mischung stehen also acht verschiedene Schriftsätze zur Verfügung:

0 = normal
1 = unterstrichen
2 = fett
3 = unterstrichen und fett
4 = schräg
5 = unterstrichen und schräg
6 = fett und schräg
7 = unterstrichen, fett und schräg (full house!)

Um an diese verschiedenen Schriftsätze heranzukommen, genügen die (schon wieder) "graphics.library"-Befehle:

```
AskSoftStyle&
SetSoftStyle&
```

Wieder haben wir einen SUB-Befehl geschrieben, den Sie so aufrufen (sollten):

```
Style x

x = 0-7
```

Sie können übrigens sämtliche von uns vorgestellten SUB-Befehle problemlos in einem Programm zusammen benutzen. So wäre beispielsweise der Einsatz verschiedener Zeichensätze in Verbindung mit diesen SoftStyles sehr interessant. Sie sollten allerdings ein und dieselbe Funktion auch nur einmal mit DECLARE FUNCTION deklarieren, um den Computer nicht restlos zu verwirren. Auch ein einziger LIBRARY-Befehl (pro Library) reicht vollends aus.

Programm-Größe: 380 Bytes
Bemerkungen: "graphics.bmap" muß sich auf Disk befinden.

```
DECLARE FUNCTION AskSoftStyle& LIBRARY
DECLARE FUNCTION SetSoftStyle& LIBRARY

LIBRARY "graphics.library"

FOR st%=0 TO 7
  style st%
  PRINT "Hello"
  style 0
  LOCATE st%+1,20
  PRINT "SoftStyle Nr. ";st%
NEXT st%

LIBRARY CLOSE
END


SUB style(style%) STATIC

bits&=AskSoftStyle&(WINDOW(8))
newStyle&=SetSoftStyle&(WINDOW(8),style%,bits&)

END SUB
```

Allgemeine Programm-Informationen: Siehe "ROMFont-Befehl"

Variablen:

loop%:	Schleife
style%:	SUB-Variable, enthält den ersehnten Style (0-7, siehe Definitionen oben)
bits&:	die von AskSoftStyle& zurückgelieferten Style-Bits
newStyle&:	der fertige neue Style

Sowohl AskSoftStyle als auch SetSoftStyle liefern Werte an AmigaBASIC zurück und müssen daher als Funktion deklariert werden. Die "graphics.library" wird geöffnet, und eine kleine Schleife führt die gesamte Pracht der neuen "Zeichensätze" vor. Danach wird die Library wieder geschlossen und das Programm endet.

Der SUB-Befehl Style ist schnell implementiert: Die Funktion AskSoftStyles& liefert die "Style-Bits" des momentan geöffneten Fonts. Diese Bits können später algorithmisch verändert werden. Anschließend wird die gewünschte Veränderung durch SetSoftStyle vorgenommen, wobei die zuvor gewonnenen Style-Bits wieder eingesetzt werden.

2.4.3 Disk-Residente Zeichensätze (Fonts)

Erwähnt hatten wir sie ja schon und damit vielleicht einigen Lesern schon den Mund wässerig gemacht. Ja, es funktioniert. Auch in AmigaBASIC kann mit den vielen netten disk-residenten Zeichensätzen herumgespielt werden, wie das im Notepad so wunderbar geht. Wieder kann man sich nicht auf AmigaBASIC selbst, sondern den leistungsstarken LIBRARY-Befehl verlassen.

Wir müssen diesmal neben der "graphics.library" eine weitere öffnen, die auf den Namen "diskfont.library" hört und deren alleiniger Zweck die Verwaltung der disk-residenten Zeichensätze ist. Sie enthält dazu nur zwei Befehle:

```
AvailFonts
OpenDiskFont
```

Aber das genügt uns schon. AvailFonts sucht im Zweifelsfall alle Disk-Fonts zusammen, die es finden kann und macht eine Liste daraus. Das ist zwar recht nett, soll uns im Moment aber nicht stören. Wir wollen so schnell wie möglich einen neuen Zeichensatz eröffnen. Dazu wieder einer unserer legendären SUB-Befehle:

```
DiskFont "name",Höhe%
```

name = Name des gewünschten Fonts, z.B. "sapphire".
Höhe% = Höhe des Fonts in Bildschirmpunkten (pixel).

Programm-Größe: 494 Bytes

Bemerkungen: "graphics.bmap" und "diskfont.bmap" müssen sich auf Disk befinden. Außerdem muß die "Workbench"-Diskette in Griffnähe gehalten werden oder sich - falls vorhanden - im zweiten Laufwerk befinden.

```
DECLARE FUNCTION OpenDiskFont& LIBRARY

LIBRARY "graphics.library"
LIBRARY "diskfont.library"

test:
altFont&=PEEKL(WINDOW(8)+52)   'im Rastport des Windows
                               'befindet sich ab Adresse
                               '52 der Zeiger auf die
                               'augenblickliche Font.
DiskFont "sapphire",19
LOCATE 5,5
PRINT "Dies ist Sapphire 19!"
LOCATE 8,1
INPUT "Zurueck zum Alten (j/n) ";jn$
IF jn$="j" THEN
  CALL SetFont&(WINDOW(8),altFont&) 'HIER wird der Zeichen-
                               'satz wieder eingeschaltet,
                               'der VOR Programmstart aktiv
                               'war. Falls das SAPPHIRE ge-
                               'wesen war, aendert sich na-
                               'tuerlich nichts; dort ROMFont
  -
                               'Befehl verwenden.
END IF

LIBRARY CLOSE
END


SUB DiskFont(font$,height%) STATIC

SHARED strFont&
prefs%=96
font0$=font$+".font"+CHR$(0)
IF strFont&<>0 THEN CALL CloseFont(strFont&)
textAttr&(0)=SADD(font0$)
textAttr&(1)=height%*2^16+prefs%
strFont&=OpenDiskFont&(VARPTR(textAttr&(0)))
IF strFont&<>0 THEN CALL SetFont (WINDOW(8),strFont&)

END SUB
```

Allgemeine Programm-Informationen: Siehe "ROMFont-Befehl"

Variablen-Feld:

textAttr&():	textAttr-Struktur, siehe ROMFont-Befehl für nähere Informationen.

Variablen:

font$:	SUB-Variable, Name des gewünschten Fonts.
height%:	SUB-Variable, Höhe des Fonts in Bildschirmpunkten.
prefs%:	SUB-Variable, Preference-Bits.
strFont&:	Zeiger auf die Verwaltung dieses Font.

Programmbeschreibung:

Da der OpenDiskFont&-Befehl eine Funktion ist, muß er als solche deklariert werden. Die beiden nötigen Libraries werden geöffnet (wobei die erste den SetFont-Befehl zum Einschalten des Fonts enthält, während die zweite den nötigen Ladebefehl OpenDiskFont& zur Verfügung stellt.

Wie auch bei der Befehlsimplementation ROMFont muß zunächst eine textAttr-Struktur ausgefüllt werden (für nähere Informationen siehe ROMFont-Befehl). In der Struktur wird neben der Zeichenhöhe auch das Feld prefs% eingefüllt, da es sich bei den Disk-Fonts ausnahmslos um Proportionalzeichensätze handelt (jedes Zeichen hat seine individuelle Breite. Das sieht super aus, ist aber schwer zu verwalten. Der Amiga macht das

natürlich mit links). Anstatt nun, wie bei ROMFont, OpenFont zu benutzen, muß OpenDiskFont gewählt werden, denn der gewünschte Font befindet sich ja noch nicht im Speicher des Rechners. Anschließend wird wieder nach gewohntem Prinzip verfahren, und der Font wird geöffnet (sofern er vorhanden ist).

Dazu gleich noch ein wenig Detail-Information: Der Amiga sucht systematisch nach dem gewünschten Font. Falls er jedoch keinen Font finden kann, auf den alle Beschreibungen (Name, Höhe und Preferences) genau zutreffen, so nimmt er den dies am besten erfüllenden Font. Und zwar wird dazu nach Möglichkeit eines gleichen Namens gefunden. Existieren solche Fonts, dann wird der Font mit der ähnlichsten Höhe gewählt, danach Style und dann Preferences verglichen. So bekommen Sie immer ein gutes Ergebnis, selbst wenn mal Ihr Lieblings-Font nicht da sein sollte.

Noch etwas zum Programm-Ablauf. Nach dem Starten des Programms benötigt es die Disk-Fonts, die sich normalerweise im Directory "Fonts" auf der Workbench-Disk befinden. Sie müssen also entweder die Diskette wechseln (wenn Sie nur ein Drive haben), die Workbench ins andere Drive legen oder ein eigenes Font-Directory auf Ihrer Disk einrichten.

Unseren Informationen zufolge stehen Ihnen diese Disk-Fonts zur Verfügung:

Font-Name	Höhe
ruby	8
ruby	12
opal	11
sapphire	14
sapphire	15
sapphire	18
sapphire	19

```
diamond     12

garnet       9
garnet      16

emerald     20
```

Es ist möglich und wahrscheinlich, daß diese Liste sowohl von Commodore als auch privaten Anbietern noch erweitert wird.

Zusammen mit den beiden ROM-(WOM?!)-Fonts stehen Ihnen also 13 völlig verschiedene Zeichensätze zur Verfügung, die jeweils auf 4 verschiedene Arten darstellbar sind (siehe Style-Befehl). Nimmt man den Outline-Befehl noch dazu, dann stehen Ihnen jetzt 13*5 = 65 verschiedene Zeichensätze zur Programmier-Verfügung. Na, wenn das keine Abwechslung ist!

2.4.4 Automatische Lagerbestandsaufnahme: AvailFonts

Nachdem der Druck etwas nachgelassen hat (die Disk-Fonts arbeiten doch wohl ordnungsgemäß?), können wir uns an den zweiten Befehl der "diskfont.library" machen. Er ist auch ganz nützlich, vor allem dann, wenn man eine gewisse Ordnung halten und einen Überblick über die bestehenden Disk-Fonts nicht verlieren möchte.

Hier der Befehl:

```
error=AvailFonts&(buffer,bufBytes,types)
```

error: Buffer war zu klein, error enthält die Anzahl der benötigten Extra-Bytes.

buffer: Zeiger auf einen freien Speicherbereich, der sich als Buffer einrichten ließe.

bufBytes: Maximale Größe des Buffers in Bytes, damit AvailFonts nicht gleich Amigas gesamtes Speichersystem überschwemmt.

types: Suche nach Disk- oder WOM-Zeichensätzen: 1=Disk Only, 2=Memory Only, 3=beides.

Programm-Größe: 1621 Bytes
Bemerkungen: "graphics.bmap", "exec.bmap" und "diskfont.bmap" müssen sich auf Disk befinden. Workbench-Disk wird auch gebraucht.

```
DECLARE FUNCTION AvailFonts& LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION OpenDiskFont& LIBRARY
DECLARE FUNCTION OpenFont& LIBRARY

LIBRARY "diskfont.library"
LIBRARY "exec.library"
LIBRARY "graphics.library"

var:
maxFonts%=20
DIM SHARED co(2)
DIM SHARED font$(maxFonts%,2)
DIM SHARED ySize%(maxFonts%,2)
DIM SHARED style%(maxFonts%,2)
DIM SHARED prefs%(maxFonts%,2)

MakeFontList

PRINT "RAM-Fonts"
PRINT
Out=1
GOSUB Out
PRINT
PRINT "Disk-Fonts"
PRINT
Out=2
GOSUB Out

LIBRARY CLOSE
END


Out:
FOR loop1%=1 TO co(Out)
  PRINT "Font-Name> ";font$(loop1%,Out)
  PRINT "Hoehe    > ";ySize%(loop1%,Out)
  PRINT "StyleBits> ";style%(loop1%,Out)
  PRINT "Prefs    > ";prefs%(loop1%,Out)
  PRINT
```

```
NEXT loop1%
RETURN

SUB MakeFontList STATIC

SHARED buffer&,result$,pointer

opt&=2^0+2^16
buffer&=AllocMem&(1000,opt&)
bf&=buffer&
IF buffer&=0 THEN
  ERROR 7
END IF
e&=AvailFonts&(buffer&,1000,3)
IF e&<>0 THEN
  ERROR 7
END IF
elements%=PEEKW(buffer&)
buffer&=buffer&+2
FOR loop&=1 TO elements%
  type%=PEEKW(buffer&)
  co(type%)=co(type%)+1
  pointer&=PEEKL(buffer&+2)
  getName pointer&
  font$(co(type%),type%)=result$
  ySize%(co(type%),type%)=PEEKW(buffer&+6)
  style%(co(type%),type%)=PEEK(buffer&+8)
  prefs%(co(type%),type%)=PEEK(buffer&+9)
  buffer&=buffer&+10
NEXT loop&
CALL FreeMem(bf&,1000)

END SUB

SUB getName(add&) STATIC

SHARED result$
result$=""

readloop:
```

```
value&=PEEK(add&)
IF value&<>0 THEN
  result$=result$+CHR$(value&)
  add&=add&+1
  GOTO readloop
END IF

END SUB
```

Allgemeine Programm-Informationen: Siehe "ROMFont-Befehl"

Variablen-Felder:

co( ):	Flag für Disk(1) und Memory(2)
font$( ):	Name der gefundenen Fonts
ySize%( ):	Höhe des Fonts
style%( ):	Style-Bits des Fonts
prefs%( ):	Preference-Bits der Fonts
textAttr&( ):	TextAttr-Struktur

Variablen:

maxFonts%:	Maximale Anzahl von Fonts pro Menü-Punkt. Im Moment =20, muß in manchen Fällen erhöht werden.
result$:	Übergabe des gefundenen Font-Namens
pointer:	Zeiger auf einen Font-Namen im Speicher
e&:	SUB1, error-Variable der AvailFonts-Funktion
type%:	SUB1, gefundener Typ: 1=Disk, 2=Memory
add&:	SUB2, Anfangsadresse des gesuchten Strings

Anmerkung:

Insbesondere wenn maxFonts% erhöht wird, kann es zu Speicherplatz-Schwierigkeiten kommen. Falls Sie eine Speichererweiterung besitzen, dann können Sie durch:

```
CLEAR ,30000
```

den nötigen Speicherplatz sicherstellen.

Programmbeschreibung:

Zunächst werden sämtliche Befehle, die einen Wert an BASIC zurückliefern, als Funktion deklariert. Dann werden die drei Libraries "diskfont", "graphics" und "exec" eröffnet. Das Programm setzt in maxFonts% einen Default-Wert von 20 Fonts pro Menü-Punkt. Sie können diesen Wert bei Bedarf natürlich erhöhen, aber Amigas ohne Speichererweiterung könnten das schon mal übelnehmen. Die nötigen Variablenfelder werden dimensioniert und als SHARED deklariert, damit die beiden SUB-Funktionen sie auch benutzen können. Anschließend wird die SUB-Funktion MakeFontList aufgerufen, die alle im System rumfliegenden Fonts zusammenkratzt und der Herkunft nach in die Variablen-Felder steckt.

Die SUB-Funktion MakeFontList macht zunächst einmal die Variablen buffer&, result$ und pointer der Außenwelt zugänglich. buffer& dient nur Kontrollzwecken, während die anderen beiden Variablen an ein zweites SUB-Programm übergeben werden müssen. Mit Hilfe des Exec-Befehls AllocMem wird ein 1000-Bytes großer Buffer lokalisiert (Er ist so dimensioniert, daß er wirklich unter allen Härtebedingungen durchkommen sollte. Für mehr Information über Memory-Handling schauen Sie doch einfach mal ins gleichnamige Kapitel!). Falls der buffer& sich nicht meldet, wird eine entsprechende Notfall-Meldung kreiert, bevor sich das Programm mit einem OUT OF MEMORY Error abmeldet. Ist jedoch alles gut gegangen, dann wird AvailFonts aufgerufen. Aber auch hier kann es passieren, daß nicht genügend Speicher zur Verfügung steht, also unter extre-

men Härtebedingungen kann es auch hier zu einem OUT OF MEMORY Error kommen. Sind diese Klippen aber erst einmal umschifft, dann liegt ruhige See voraus. Der von AvailFonts gefüllte Buffer wird nun ausgelesen. Dazu ist es notwendig, den Aufbau dieses Buffers zu verstehen. Die ersten beiden Bytes legen die Anzahl der Einträge fest, die in elements% gespeichert werden. Danach folgen die Einträge, die das folgende C-Format haben:

```
UWORD af Type (1=disk, 2=memory)
struct TextAttr
```

(unsere beliebte textAttr-Struktur, siehe ROMFont-Befehl)

Die einzelnen Felder werden mit den Buffer-Werten gefüllt, wobei die SUB-Routine GetName den Font-Namen aus dem Speicher liest.

Ist der Buffer ausgeschlachtet, besteht eigentlich kein Grund mehr, ihn aufrecht zu erhalten. Daher werden die wertvollen 1000 Bytes durch FreeMem wieder freigegeben.

Die SUB-Routine GetName sucht so lange ASCII-Codes aus dem Speicher, bis eine 0 erreicht ist.

Zum Programm:

Nach dem Starten und Laden der ".bmap"-Files benötigt das Programm die Workbench. Legen Sie diese ins andere Laufwerk, oder wechseln Sie die Disketten. Nun werden Sie ein schabendes Geräusch hören, was aber entgegen dem ersten Eindruck völlig ungefährlich ist: AvailFonts liest nämlich auf recht ohrenbetäubende Weise alle auf der Workbench zu findenden Fonts ein. Ein paar Sekunden später ist der Spuk vorbei, und Sie können durch Mausdruck die Liste der gefundenen Fonts abklappern, die nach Memory und Disk getrennt ausgegeben werden

Auf einigen Workbench Disketten existiert übrigens eine Phantom-Font namens Ruby/12 (neben dem offiziellen).

Noch eine Bemerkung zum Programm: Sie können es natürlich modifizieren, indem Sie ein eigenes kleines Programm an Stelle der Ausgabeschleife einbauen (dann hat es wenigstens einen Sinn). Dazu gibt's auch gleich von uns noch ein paar Anregungen!

2.4.5 Transparente drucken

Und hier gleich ein Einsatz der AvailFonts-Funktion wie aus dem Leben gegriffen: Die Disk-Fonts machen sich ja recht gut auf dem Bildschirm. Aber man kann eigentlich noch eine ganze Menge mehr mit ihnen machen. Jeder Buchstabe eines Fonts ist dort ja als Punktraster definiert worden. Was hindert uns daran, diese Definition unsererseits auszulesen, die Matrix zu vergrößern und als Transparent auf einen Drucker auszugeben? Natürlich nichts!

Programm-Größe: 4775 Bytes
Bemerkung: "exec.bmap", "dos.bmap" sowie "diskfont.bmap" müssen sich auf Disk befinden. Workbench-Disk parat halten.

```
'transparentdrucker
'modified 13.Nov.86/Workbench Version 1.2

DECLARE FUNCTION AvailFonts& LIBRARY
DECLARE FUNCTION DosOpen& LIBRARY
DECLARE FUNCTION DosRead& LIBRARY
DECLARE FUNCTION DosSeek& LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY

LIBRARY "exec.library"
LIBRARY "dos.library"
LIBRARY "diskfont.library"

PRINT "*** TRANSPARENT-DRUCKER ***"
PRINT "  Initialisierung laeuft!"

maxFonts%=20
DIM ret$(400)
DIM SHARED font$(maxFonts%)
DIM SHARED hgt%(maxFonts%)
hoehe=1    'Default-Werte
weite=1
pl$="*"    'gesetzter Punkt
upl$=" "   'Leerzeichen
```

```
main:

GOSUB getMemory
GOSUB preparePrinter
GOSUB initMenu

WHILE fl=0

 MenuId=MENU(0)
 MenuItem=MENU(1)
 MENU OFF
 IF MenuItem=1 AND MenuId=3 THEN fl=1
 IF MenuId=1 THEN
  in$=font$(MenuItem)
  hgt%=hgt%(MenuItem)
 END IF
 IF MenuId=2 THEN
  IF MenuItem=1 THEN
    GOSUB printText
  ELSEIF MenuItem=2 THEN
    GOSUB ChangeParameters
  END IF
 END IF
 LOCATE 1,1
 weite$=RIGHT$(STR$(weite),LEN(STR$(weite))-1)
 hoehe$=RIGHT$(STR$(hoehe),LEN(STR$(hoehe))-1)
 PRINT "FONT: ";in$;"  HOEHE: x";hoehe$;" = ";hoehe*hgt%;"CH
  AR; WEITE: x";weite$;STRING$(40," ")
 PRINT "MENU-gesteuerter Transparentdrucker"
 MENU ON

WEND

CALL FreeMem(b&,8000)
CLOSE 1

LIBRARY CLOSE
END


ChangeParameters:

PRINT:PRINT "Vergroesserung:"
PRINT
LINE INPUT "Hoehe>  1:";hoehe$:hoehe=VAL(hoehe$)
LINE INPUT "Weite>  1:";weite$:weite=VAL(weite$)
CLS
RETURN

printText:

ReadFile in$,8000&,0&,b&
GOSUB getVar
PRINT "FONT> ";in$
PRINT "File ID> ";HEX$(PEEKW(b&+50))
```

```
diff%=hiChar%-loChar%
LINE INPUT "TEXT> ";a$
PRINT:PRINT "* printing *"
PRINT
FOR x=1 TO LEN(a$)
  b$=MID$(a$,x,1)
  PRINT b$;
  loop%=ASC(b$)-loChar%
  GOSUB printIt
NEXT x
CLS
RETURN

getVar:

fontEnd%=PEEKW(b&+108)
ySize%=PEEKW(b&+110)
xSize%=PEEKW(b&+114)
baseLine%=PEEKW(b&+116)
loChar%=PEEK(b&+122)
hiChar%=PEEK(b&+123)
charData&=(PEEKL(b&+124)+32)+b&
Modulo%=PEEKW(b&+128)
fontLoc&=(PEEKL(b&+130)+32)+b&
fontSpace&=(PEEKL(b&+134)+32)+b&
fontKern&=(PEEKL(b&+138)+32)+b&

RETURN

initMenu:

MakeMenuList
MENU 1,0,1,"Fonts"
MENU 2,0,1,"Funktionen"
MENU 3,0,1,"Beenden"
MENU 2,1,1,"Text ausdrucken"
MENU 2,2,1,"Parameter aendern"
MENU 3,1,1,"OK. Das war's"
FOR loop%=1 TO co
  MENU 1,loop%,1,RIGHT$(font$(loop%),LEN(font$(loop%))-5)
```

```
NEXT loop%
MENU ON
RETURN

getMemory:

opt&=2^0+2^16
b&=AllocMem&(8000,opt&)
IF b&=0 THEN ERROR 7
RETURN

preparePrinter:

OPEN "prt:" FOR OUTPUT AS 1
PRINT#1,CHR$(27)+"[0z"
RETURN

printIt:

start%=PEEKW(fontLoc&+4*loop%)
wide%=PEEKW(fontLoc&+2+4*loop%)
space%=PEEKW(fontSpace&+2*loop%)
kern%=PEEKW(fontKern&+2*loop%)
FOR loop2%=0 TO ySize%-1
  trstart&=start%+(loop2%*Modulo%)*8
  transvert trstart&,wide%
  FOR hgt=1 TO hoehe
    ret$(counter)=ret$
    counter=counter+1
  NEXT hgt
  STOP
NEXT loop2%
sp%=space%-wide%-kern%
FOR x%=1 TO kern%
  PRINT#1,""
NEXT x%
FOR x%=1 TO LEN(ret$)
  FOR y%=counter TO 0 STEP -1
    PRINT#1,MID$(ret$(y%),x%,1);
  NEXT y%
```

```
  PRINT#1,""
NEXT x%
FOR x%=0 TO sp%
  PRINT#1,""
NEXT x%
counter=0
RETURN

Request:

LINE INPUT "FONT> ";in$
RETURN


'-------------------------------------

SUB MakeMenuList STATIC

SHARED buffer&,result$,pointer,co

opt&=2^0+2^16
buffer&=AllocMem&(1000,opt&)
IF buffer&=0 THEN ERROR 7
bf&=buffer&
e&=AvailFonts&(buffer&,1000,2)
IF e&<>0 THEN
  PRINT "We need ";STR$(e&);"more bytes!"
  ERROR 7
END IF
elements%=PEEKW(buffer&)
buffer&=buffer&+2
FOR loop&=1 TO elements%
  type%=PEEKW(buffer&)
  co=co+1
  pointer&=PEEKL(buffer&+2)
  getName pointer&
  font$(co)="Fonts:"+LEFT$(result$,LEN(result$)-5)
  hgt%(co)=PEEKW(buffer&+6)
  hgt$=STR$(hgt%(co))
  font$(co)=font$(co)+"/"+RIGHT$(hgt$,LEN(hgt$)-1)
```

```
  buffer&=buffer&+10
NEXT loop&
CALL FreeMem(bf&,1000)

END SUB


SUB getName(add&) STATIC

SHARED result$
result$=""

readloop:
value&=PEEK(add&)
IF value&<>0 THEN
  result$=result$+CHR$(value&)
  add&=add&+1
  GOTO readloop
END IF

END SUB


SUB transvert (bit&,many%) STATIC

SHARED charData&,ret$,weite,pl$,upl$

plot$=STRING$(weite,pl$)
unplot$=STRING$(weite,upl$)
ret$=""
byteOffset%=INT(bit&/8)
bits%=bit&-(INT(bit&/8)*8)
IF bits%=0 THEN byteOffset%=byteOffset%-1
FOR check%=bits% TO bits%+many%-1
  expo%=(check% MOD 8)
  IF expo%=0 THEN byteOffset%=byteOffset%+1
  check&=PEEK(charData&+byteOffset%)
  IF (check& AND 2^(7-expo%))<>0 THEN
    ret$=ret$+plot$
  ELSE
```

```
    ret$=ret$+unplot$
  END IF
NEXT check%

END SUB


SUB ReadFile(file$,bytes&,position&,buffer&) STATIC

SHARED gelesen&
ModeOldFile%=1005
offsetBeginning%=-1
file$=file$+CHR$(0)
handle&=DosOpen&(SADD(file$),ModeOldFile%)
IF handle&=0 THEN
  PRINT "File gibt es nicht"
  fl=1
END IF
oldPos&=DosSeek&(handle&,position&,offsetBeginning%)
gelesen&=DosRead&(handle&,buffer&,bytes&)
CALL DosClose(handle&)

END SUB
```

Variablenfelder:

ret$:	Vergrößerte Punktematrix
font$:	Font-Name
hgt%:	Font-Höhe

Variablen:

maxFonts%:	Felderdimensionierung
hoehe:	Vergrößerung Y-Richtung
weite:	Vergrößerung X-Richtung
pl$:	gesetzter Punkt
upl$:	nicht gesetzter Punkt
MenuId:	Menü-Hauptpunkt
MenuItem:	Menü-Unterpunkt
in$:	Arbeitsvar., Font-Name
hgt%:	Arbeitsvar., Font-Höhe
weite$:	Formatierte Weitenangabe
hoehe$:	s.o., Höhenangabe
diff%:	Anzahl definierter Zeichen
a$:	Text
b$:	Textzeichen
loop%:	Zeichencode
fontEnd%:	Zeiger auf Font-Ende
ySize%:	Font-Höhe
xSize%:	Font-Weite
baseLine%:	Fußlinie
loChar%:	ASCII; untere Definitionsgrenze
hiChar%:	ASCII; obere Def.Grenze
charData&:	Zeiger auf Zeichendaten

Modulo%:	Speicheroffset
fontLoc%:	Definitionsentschlüsselung
fontSpace&:	Zeiger auf Font-Weitentabelle
fontKern&:	Zeiger auf Font-Kern-Tabelle
opt&:	Speicher-Optionen
b&:	Buffer-Adresse
start%:	Datenanfang lokal
wide%:	Datenweite lokal
space%:	Zeichenweite lokal
kern%:	Kern lokal
trstart%:	Aktueller Speicherstart
elements%:	Anzahl DiskFonts
buffer&:	Zweiter Speicher
byteOffset%:	Offset in Bytes
bits%:	Übriger Offset in Bits
expo%:	wenn 0: nächstes Byte muß her

Programmbeschreibung:

Dieses Programm richtet zunächst einen 8K-Datenbuffer via getMemory ein. Anschließend wird der Drucker durch preparePrinter auf 6"/line geschaltet (engere Zeilen). Danach wird das Menü initialisiert. Dazu wird MakeMenuList aufgerufen, welches alle verfügbaren Disk-Fonts in Variablenfeldern speichert. Diese werden ausgelesen und in den ersten Menüpunkt transferiert.

Nun können beliebige Fonts durch Anklicken im Menü selektiert werden. Die Funktion ChangeParameters läßt die Vergrößerung der Fonts in X- und Y-Richtung zu. Es sollte lediglich beachtet werden, daß die Y-Vergrößerung * Höhe des Fonts nicht die Anzahl der vom Drucker horizontal druckbaren Zeichen überschreitet. Ausprobieren hilft.

Der Menüpunkt "Text ausdrucken" fragt nach dem Drucktext. Anschließend wird der entsprechende Font in den Datenbuffer geladen. Dies geschieht durch ReadFile, da AmigaBASICs OPEN die Nullen verschlucken würde.

Das bitpacked Image eines jeden Zeichens wird ausgerechnet, entsprechend vergrößert und um 90 Grad gedreht auf den Drukker ausgegeben.

2.5 Intuition und mehr

Intuition, eine weitere Library des Amiga, ist hauptsächlich für Windows, Screens und Requester verantwortlich. AmigaBASIC macht davon auch eifrig gebrauch, wenn es um ebendiese Funktion geht, aber voll ausgenutzt wird sie noch nicht. Wieder sind ein paar unserer glorreichen SUB-Routinen nötig, um auch Sie in den Genuß der vollen Möglichkeiten zu bringen.

2.5.1 Ein Automatik-Requester

Na, wie oft kommt es vor, daß ein Programm eine Frage an den Benutzer stellen muß? Zehnmal? Hundertmal? Das fängt doch schon bei Standard-Fragen wie "Bitte Disk wechseln" an und reicht bis zu Data Beckers speziellen "Sind Sie sicher?"-Warnungen. In einem Punkte stimmen die Fragen, die man auch ins schöne Englisch übersetzen und Requests nennen könnte, jedoch überein: Normalerweise benötigt das Programm nur eine sogenannte "boolean" Antwort. Ein einfaches "Ja" oder "Nein" reicht also schon aus.

Und genau dafür hat die Intuition schon etwas vorrätig, das auch BASIC-Programmen den gewissen Touch der Professionalität geben kann: Einen Auto-Requester. Er wird mit dem folgenden Format aufgerufen:

```
res&=AutoRequest(WINDOW(7),bodyT,yesT,noT,yesF,noF,Weite, Höhe)

bodyT   : Zeiger auf IntuiText-Struktur mit allgemeinem Text
yesT    : Zeiger auf IntuiText-Struktur mit Text für das Ja-Feld
noT     : Zeiger auf IntuiText-Struktur mit Text für das Mitnichten-Feld
yesF    : Flags für Ja
```

```
noF     : Flags für Nein
Weite   : Weite des Requesters (0-640)
Höhe    : Höhe des Requesters (0-200, bzw. 0-400)
```

Es gilt jedoch noch eine schreckliche Hürde zu überspringen. Als die Techniker bei Amiga den Rechner zusammenbauten, schielten Sie sicherlich ununterbrochen mit einem Auge nach links auf einen kleinen Tisch, auf dem sich auf einer Diskette ein C-Compiler befand. Jedenfalls muß vor Gebrauch des AutoRequests für jeden Text (und davon gibt es mindestens drei) eine IntuiText-Struktur ausgefüllt werden. Von C aus ist es ein Kinderspiel, aber in BASIC wird schon einiges extensives Poken nötig. Deshalb finden Sie auch gleich eine alles sooo viel einfacher machende SUB-Routine, die folgendermaßen aufgerufen wird:

```
Request weite%,höhe%,lines%

weite%   : (0-640)
höhe%    : (0-200/0-400)
lines%   : Haupttext-Zeilen (-1)
```

Programm-Größe: 1751 Bytes

Bemerkungen: "intuition.bmap" muß sich auf Disk befinden.

```
' AutoRequester

DECLARE FUNCTION AutoRequest& LIBRARY
DECLARE FUNCTION AllocRemember& LIBRARY

LIBRARY "intuition.library"

var:
DIM SHARED body$(4)
body$(0)="Dies ist ein Requester, mit dem man saeumige User"

body$(1)="bestraft. Text Nummero 2 passt noch locker mit rei
  n!"
yes$="Jawoll, dem ist wohl so!"
no$="Aber auf gar keinen Fall."

main:

Request 600,80,1 '1: body-Text 0 und 1
                 '2: wuerde body-Text 0, 1 und 2 in den Requ
  ester nehmen
```

```
IF res&=1 THEN
  PRINT "Na, wenigstens einer, der auf mich hoert!"
ELSEIF res&=0 THEN
  PRINT "Da haben Sie auch wieder Recht, eigentlich zu schad
  e drum!"
END IF

LIBRARY CLOSE
END

SUB Request(wid%,hi%,bt%) STATIC

SHARED add&,st$,res&,body$,yes$,no$,offs%

opt&=2^0+2^16
req&=AllocRemember&(0,400,opt&)
IF req&=0 THEN ERROR 7
add&=req&

t1&=add&
FOR loop2=0 TO bt%-1
st$=body$(loop2)
MakeHeader add&,st$,1,5,offs%+3
offs%=offs%+8
NEXT loop2
st$=body$(bt%)
MakeHeader add&,st$,0,5,offs%+3
st$=yes$
t2&=add&
MakeHeader add&,st$,0,5,3
st$=no$
t3&=add&
MakeHeader add&,st$,0,5,3

res&=AutoRequest&(WINDOW(7),t1&,t2&,t3&,0,0,wid%,hi%)
CALL FreeRemember(0,-1)

END SUB
```

```
SUB MakeHeader(ptr&,text$,md%,le%,te%) STATIC

SHARED add&
text$=text$+CHR$(0)

POKE ptr&,1              'Front Pen
POKE ptr&+1,0            'Back Pen
POKE ptr&+2,2            'JAM1
POKEW ptr&+4,le%         'relative left edge offset
POKEW ptr&+6,te%         'relative top edge offset
POKEL ptr&+8,0           'default font
POKEL ptr&+12,SADD(text$) 'text
IF md%=0 THEN
  POKEL ptr&+16,0          'no cont
ELSE
  POKEL ptr&+16,ptr&+20
END IF

add&=ptr&+20

END SUB
```

Variablen-Felder:

body$: Enthält die Zeilen des Text-Körpers

Variablen:

yes$:	Enthält Text für's JA-Feld
no$:	Enthält Text für's NEIN-Feld
res&:	Vom Requester zurückgeliefertes Resultat: 1=Ja, 0=Nein
wid%:	Weite des Requesters in Bildschirm-Punkten
hi%:	Höhe des Requesters in Bildschirm-Punkten
bt%:	Nummer des letzten definierten Elementes in body$
opt&:	gewünschte Memory-Art
req&:	Anfangsadresse des neu angelegten Buffers
add&:	Buffer-Zähler
t1&:	Zeiger auf body-IntuiText
t2&:	Zeiger auf yes-IntuiText
t3&:	Zeiger auf no-IntuiText
st$:	Übergabe-Variable für MakeHeader
offs%:	Zeilen-Offset für Body-Text
text$:	Gerade bearbeiteter Text
ptr&:	MakeHeader-Zeiger

Zum Aufruf der Routine:

Vor dem Aufruf der Requester-Routine müssen die folgenden Variablen initialisiert werden:

yes$:	Eine Zeile Text, die in das Ja-Feld eingebaut wird. Je kürzer desto besser.
no$:	Wie zes$, jedoch für's Nein-Feld

Für den Text-Körper können Sie mehr als eine Zeile verwendet. Legen Sie die Zeilen bitte der Reihe nach in das Variablen-Feld body$:

```
body$(0)="Zeile 1"
body$(1)="Zeile 2"
(...)
body$(5)="Zeile 6"
```

Beim Aufruf der Request-Funktion geben Sie bitte die Weite und Höhe des Requesters an. Außerdem muß die Nummer des letzten definierten Elementes in body$ angegeben werden (im obigen Beispiel =5).

Achtung: Für die Proportionen Ihres Requesters müssen Sie selbst sorgen! Zwar wird Intuition immer einen genügend großen Rahmen für die Ja- und Nein-Felder zeichnen, aber wenn der gesamte Requester zu klein ist, um Ihre Texte aufzunehmen, dann kommt es zu recht eigentümlichen Ergebnissen (Mode=COMPLEMENT...).

Während der Requester "draußen" ist, hält AmigaBASIC natürlich im Programm-Ablauf inne. Es geht erst weiter, wenn mit der Maus entweder das Ja- oder das Nein-Feld angeklickt wird. In der Variablen res& wird das Ergebnis der Frage zurückgeliefert: 1=Ja, 0=Nein. So können Sie auf die Antwort eingehen (ansonsten wäre der ganze Requester ohnehin sinnlos).

Programm-Beschreibung:

Die beiden benötigten Intuition-Funktionen werden deklariert und die Library geöffnet. Das Feld body$ wird eingerichtet und als SHARED deklariert, damit die Sub-Routine Request auch an die Textzeilen herankommt. Im Moment hat dieses Feld 4 Elemente, es können also maximal 5 Zeilen Haupt-Text ausgegeben werden. Es liegt jedoch ganz bei Ihnen, diesen Wert zu erhöhen.

Die nötigen Textvariablen werden für einen Testdurchlauf mehr oder weniger geistreich bestückt, und dann wird der Requester aufgerufen (600 Punkte breit, 80 Punkte hoch, 2 Zeilen Haupttext; 0 und 1). Anschließend wird dem Resultat zufolge geantwortet.

Die SUB-Routine Request erklärt erst einmal eine ganze Reihe Variablen für allgemeingültig, da viele an MakeHeader weitergegeben oder ans Hauptprogramm zurückgeliefert werden müssen. Ein 400 Bytes umfassendes Speicherplätzchen wird gesucht und (hoffentlich) gefunden. Anschließend wird an den Anfang

dieses Buffers von MakeHeader die IntuiText-Struktur für die erste body-Zeile gePOKEd. Ist übrigens nur eine Zeile dafür vorgesehen, dann tut sich in der Schleife loop2 gar nichts.

Die zweite SUB-Routine, MakeHeader, ist in der Lage, eine IntuiText-Struktur in den Speicher zu poken. Dazu werden ihr die gewünschte Anfangsaddresse, der Text, ein Flag und die x- und y-Positionen des Textes übergeben. Ist das Flag=1, dann stellt MakeHeader in der IntuiText-Struktur einen Link zum nächsten Block her. So können mehrere Zeilen Text im Haupttext ausgegeben werden (ist auch im Ja- und Nein-Feld möglich).

Die IntuiText-Blöcke werden in den Speicher gebracht, wobei die jeweiligen Anfangsadressen in t(1,2,3)& untergebracht werden. Nun kann die eigentliche Funktion AutoRequest aufgerufen werden, die den Requester zusammenbaut. In res& wird das Resultat zurückgemeldet. Nun muß nur noch der Buffer freigegeben werden, und Request hat seine Arbeit getan.

2.5.2 Intuition-Alarm

Das plötzliche Überschwemmen des Bildschirmes mit einer dramatisch rot-blinkenden Alarm-Meldung ist schon ein Ereignis. Oftmals ein schlechtes, aber es bleibt ein Ereignis. Das gesamte System hält den Atem an, während der Alarm - oftmals letzte Rettung vor einem System-Absturz - auf dem Bildschirm blinkt. Sie sollten aus diesem Grund einen Intuition-Alarm nur in taktisch wichtigen Situationen verwenden. In allen anderen Fällen tut ein ganz gewöhnlicher Requester genau denselben Dienst, nur fährt dem Benutzer nicht so ein Schrecken in die Knochen, und das Multi-Tasking-System kann auch frohgemut ohne Unterbrechung und lästige Wartereien arbeiten.

Der Intuition-Alarm sorgt für einen blinkenden roten Rahmen auf schwarzem Hintergrund bei der 640-Pixel-Auflösung. Er kann so hoch wie nötig sein und gegebenenfalls den gesamten Bildschirm ausfüllen. Der Alarm erscheint immer am oberen Bildschirmrand und drückt den bestehenden Screen, sofern er

noch funktionsfähig ist, nur herunter. Handelt es sich jedoch um einen unheilbaren Alarm, und das System steht kurz vor dem Absturz, dann übernimmt der Alarm automatisch den gesamten Bildschirm.

Sie als Benutzer können natürlich auch Alarm-Meldungen erzeugen, denn Ihnen stehen dieselben Mittel zur Verfügung, auf die auch das System zurückgreifen muß. Es geht von BASIC aus sogar ziemlich leicht, wenn Sie mit Kompromissen einverstanden sind:

```
ERROR 14
```

produziert einen OUT-OF-HEAP-SPACE-Error, der das Format eines Alarms aufweist.

Grundsätzlich gibt es zwei verschiedene Alarm-Typen:

DEAD-END (nix mehr zu machen) und *RECOVERY* (da können wir noch was dran drehen). Im ersten Fall handelt es sich nur um eine Meldung an den Benutzer, daß sich der Rechner im Absturz befindet, und es spielt überhaupt keine Rolle, welche der beiden Maus-Tasten Sie zur Bestätigung drücken. Anders sieht es im zweiten Fall aus, bei dem 0 für die rechte und 1 für die linke Taste zurückgemeldet wird.

Um einen Alarm zu produzieren, bedient man sich am Besten des Intuition-Befehls:

```
res=DisplayAlert(AlarmNummer, String, Höhe)
```

Bei der Alarmnummer handelt es sich um eine LONG-Variable (z.B. a&). Nur die ALERT-TYPE-Bits spielen eine Rolle.

Der String enthält die eigentliche Meldung. Er ist folgendermaßen aufgebaut:

```
16-bit x-Koordinate
8-bit y-Koordinate
```

```
ASCII-String, mit 0 abgeschlossen
Cont-Byte (0=das war's, 1=noch ein Text)
```

Die Höhe muß in Bildschirmpunkten angegeben werden und bestimmt die Größe der Alarmmeldung.

Schauen Sie sich einfach das folgende Beispielprogramm an, das einen RECOVERY-Alert produziert (wir wollen schließlich noch weitermachen!).

```
DECLARE FUNCTION DisplayAlert& LIBRARY

LIBRARY "intuition.library"

var:
alertNum&=0  'recovery alert
text$=CHR$(0)+CHR$(30)+CHR$(20)+"Hallo Jungs, dies ist eine
  dieser tollen Alarmmeldungen,"
text$=text$+CHR$(0)+CHR$(1)+CHR$(0)+CHR$(30)+CHR$(28)+"die e
  inem normalerweise das Blut in den Adern"
text$=text$+CHR$(0)+CHR$(1)+CHR$(0)+CHR$(30)+CHR$(36)+"gefri
  eren lassen. Diese hier ist aber ganz "
text$=text$+CHR$(0)+CHR$(1)+CHR$(0)+CHR$(30)+CHR$(44)+"harml
  os. "
text$=text$+CHR$(0)+CHR$(1)+CHR$(0)+CHR$(30)+CHR$(56)+"Druec
  ken Sie einfach die linke Mouse-Taste, wenn"
text$=text$+CHR$(0)+CHR$(1)+CHR$(0)+CHR$(30)+CHR$(64)+"Sie d
  ieses Buch moegen, sonst die rechte!"
text$=text$+CHR$(0)+CHR$(1)+CHR$(0)+CHR$(15)+CHR$(82)+"[abso
  lute Spitzenklasse!]                [fast ausgezeichnet]"
text$=text$+CHR$(0)
```

```
main:

res&=DisplayAlert&(alertNum&,SADD(text$),100)
IF res&=1 THEN
  PRINT "Gut, der Mann! die Frau! Sie verstehen schon, wann
  Sie ein"
  PRINT "gutes Buch in den Haenden halten. Gratuliere!"
ELSE
  PRINT "Kultur-Banause! Man sollte Sie ueber's Knie legen..
  ."
  PRINT "WER, wenn nicht wir, haette Ihnen beigebracht, wie
  man diese"
  PRINT "tollen Alarm-Meldungen produziert? Hm? Wollen wir e
  s noch"
  PRINT "einmal versuchen?"
  FOR t=1 TO 15000
  NEXT t
  RUN

  END IF
```

Die vielen CHR$-Codes dienen der Positionierung und Verbindung der einzelnen Zeilen. Näheres dazu siehe oben.

Noch einmal wollen wir darauf hinweisen, daß bei einem Alarm das gesamte System stehenbleibt. Falls Sie noch ein zweites oder drittes (BASIC-)-Programm ausgeführt hatten, dann wird auch diesem für die Dauer des Alarms der Strom abgedreht. Noch ein kleiner Scherz am Rande: Sollten Sie einmal über ein bißchen Extra-Zeit verfügen, dann setzen Sie einmal für alertNum& den Wert &H8000 ein! Sie verwandelt so den Recovery-Alert in einen Dead-End. Viel Spaß!

Übrigens: Obwohl Ihr Screen verschwunden ist, hat sich das System nicht abgemeldet. Schließlich waren Sie es durch Ihr Programm, der den DEAD-END-Alarm produziert hat. Folgerichtig gibt Intuition die Kontrolle auch wieder an Sie ab. Ihr BASIC-Programm läuft also weiter, obwohl Ihr Screen verschwunden ist. Hatten Sie zum Beispiel ein Vogelstimmen-Imitator-Programm im Speicher, dann werden auch weiterhin Vogelstimmen in der Luft liegen. Zwar gibt es Mittel und Wege, wieder an einen Screen heranzukommen, aber wozu der Umstand? Sofern Sie einen RECOVERY-Alert spezifieren, haben Sie all diese Probleme ja gar nicht.

2.5.3 Offizielle Alarm-Meldungen

Neben all diesem Unsinn kann es natürlich auch mal vorkommen, daß eine echte Alarm-Meldung dazwischenrutscht. Dann wird die Sache ernst, denn das System gibt diese Meldungen im Gegensatz zu uns nicht zur allgemeinen Belustigung aus, sondern weil ein Zustand eingetreten ist, den es nicht mehr selbst beheben kann (und was kann der Amiga nicht selbst beheben?).

Falls also eine solche Meldung auf dem Schirm auftaucht, dann drücken Sie bitte nicht blitzschnell auf eine der Maus-Tasten, sondern schlagen Sie diese Seiten auf. Würde die Alarmmeldung dem Benutzer nur sagen: "Hallo, tschüß! Ich geh' jetzt!", dann hätte man auch gleich auf sie verzichten können, und der Rechner wäre sang- und klanglos zehn Sekunden früher abgeschmiert. Jeder Intuition-Alarm führt eine sogenannte "Guru Meditation"-Nummer mit sich. Diese Nummer kann bei der Fehlersuche sehr hilfreich sein, kennt man ihre Dekodierung. Hier ist sie:

Alarm Typen:

0x80000000	Dead End
0x00000000	Recovery

Allgemeine Alarm-Ursachen:

0x00010000	Kein Speicherplatz
0x00020000	Library zerstört
0x00030000	Library konnte nicht geöffnet werden
0x00040000	Gerät konnte nicht benutzt werden
0x00050000	Resource konnte nicht geöffnet werden
0x00060000	I/O Error

Alarm-Objekte:

0x00008001	exec.library
0x00008002	grphics.library
0x00008003	layers.library
0x00008004	intuition.library

0x00008005	math.library
0x00008006	clist.library
0x00008007	dos.library
0x00008008	ram.library
0x00008009	icon.library
0x00008010	audio.device
0x00008011	console.device
0x00008012	gameport.device
0x00008013	keyboard.device
0x00008014	trackdisk.device
0x00008015	timer.device
0x00008020	cia.resource
0x00008021	disk.resource
0x00008022	miscellaneous.resource
0x00008030	Bootstrap
0x00008031	Workbench

Spezielle Alarm-Meldungen:

exec.library:

0x81000001	68000 exception vektor checksum
0x81000002	execbase checksum
0x81000003	library checksum failure
0x81000004	no memory to make library
0x81000005	corrupted memory list (häufig.)
0x81000006	no memory for interrupt servers
0x81000007	InitStruct() of an APTR-Source

graphics.library:

0x82010001	Copper display list out of memory
0x82010002	Copper instruction list out of memory
0x82000003	Copper list overload
0x82000004	Copper intermediate list overload
0x82010005	Copper list head out of memory
0x82010006	long frame out of memory
0x82010007	short frame out of memory

0x82010008	flood fill out of memory
0x82010009	no memory for TmpRas for text
0x82010010	BltBitMap out of memory

intuition.library:

0x84000001	unknown gadget type
0x84010002	no memory for create port
0x84010003	item plane allocation, no memory
0x04010004	sub allocation out of memory
0x84010005	plane allocation out of memory
0x84000006	item box top smaller than RelZero
0x84010007	no memory for open screen
0x84010008	no memory for open screen raster allocation
0x84000009	open sys screen of unknown type
0x8401000A	no memory for adding SW Gadgets
0x8401000B	no memory for open window
0x8400000C	entering intuition: bad state return
0x8400000D	bad message received by Intuition DCP
0x8400000E	weird echo causing incomprehenson
0x8400000F	Console Device couldn't be opened

dos.library:

0x07010001	no memory at startup
0x07000002	EndTask didn't end
0x07000003	Qpacket failure
0x07000004	unexpected packet received
0x07000005	Freevec failed
0x07000006	Disk Block Sequence Error
0x07000007	Bitmap corrupt
0x07000008	Key already free
0x07000009	Invalid Checksum
0x0700000A	Disk Error
0x0700000B	Key Out of Range
0x0700000C	Bad Overlay

Dies sind die wichtigsten Error-Codes. Nur den wenigsten werden Sie jedoch in Alarm-Meldungen Auge in Auge gegenüber-

stehen, da das Betriebssystem versucht, diese Fehler vorher selbst zu beheben.

2.5.4 Window-Manipulationen

Nanu, wie kommen wir denn auf dieses Thema, noch dazu an dieser Stelle? Obwohl Sie Windows vermutlich mit dem Amiga-BASIC-Befehl WINDOW öffnen und verwalten, werden sie in Wirklichkeit von Intuition produziert und überwacht. So schön und nett der WINDOW-Befehl auch sein mag, auf die Dauer wird es langweilig. Da muß doch noch mehr zu machen sein... Früher oder später wird der ambitionierte Programmierer die Hintertür entdecken, die in AmigaBASIC eingebaut wurde: Durch Aufruf der Kommandos WINDOW(7) bzw. WINDOW(8) kommt man an den Rast-Port und die Intuition-Verwaltung eines (gerade mit WINDOW OUTPUT selektierten) Windows. Der Rast-Port wurde bereits des öfteren in Verbindung mit Befehlen aus der graphics.library benutzt, die Intuition-Verwaltung ist neu.

Jedes durch Intuition geöffnete Window (im Normalfall sämtliche Windows auf dem Bildschirm, es sei denn, Sie haben Ihr eigenes Betriebssystem entwickelt) besitzt eine WINDOW-Struktur. Diese enthält zum einen wichtige Daten über das Window selbst, zum anderen bietet sie einen Link zur vorangegangenen und nachfolgenden WINDOW-Struktur. Im Folgenden wollen wir Ihnen ein paar kleine Tricks damit zeigen.

2.5.4.1 ClearMenuStrip

Die Intuition-Funktion ClearMenuStrip kann mit einem Aufruf das gesamte Menü eines Windows ausradieren. Dies funktioniert so:

```
LIBRARY "intuition.library"

CALL ClearMenuStrip(WINDOW(7)) 'Menustrip des OUTPUT Windows


'In den Normalzustand gelangt man durch Eingabe
'von:
'
'MENU RESET

LIBRARY CLOSE
END
```

Der Sinn dieser Funktion ist allerdings recht zweifelhaft: Sobald von BASIC aus ein neues Menü definiert wird, tauchen die Reste des alten wieder auf. Zum kurzfristigen Verschlucken des Menüs (um Fehlbedienung zu vermeiden oder den Benutzer zu erschrecken) ist sie jedoch recht nützlich.

2.5.4.2 Veränderung des IDCMP

Den IDCMP Intuition Direct Communications Message Port kann man für jedes Window von Intuition einrichten lassen. So können dann wichtige Informationen bezüglich des Windows, z.B. Maus-Bewegungen, Menü-Wahl etc., an ein potentielles Programm weitergegeben werden.

Genau dies geschieht im Fall von AmigaBASIC. Durch den Intuition-Befehl ModifyIDCMP läßt sich das IDCMP-Flag des Windows manipulieren. Dieses Flag bestimmt, welche aus der Fülle der zur Verfügung stehenden Informationen an das Programm übergeben werden. Durch systematisches Verändern dieses Flags läßt sich schon was machen:

```
LIBRARY "intuition.library"

fl%=&H36E

CALL ModifyIDCMP(WINDOW(7),fl%)

PRINT "Um wieder zurueck in den Normalzustand zu kommen,"
PRINT "bitte im Menu 'CONTINUE' anwaehlen!"

STOP
CALL ModifyIDCMP(WINDOW(7),&H76E)
```

fl% ist dabei der neue Inhalt des Flags. Hier ein paar Werte und deren Effekte:

&H76E normal

&H766 MOUSEBUTTONS wurde ausgeschaltet. Der BASIC-Befehl MOUSE(0) wartet nun ewig auf ein Zeichen von der Maus. Auch das Umschalten zwischen LIST und BASIC wird erschwert.

&H66E MENUPICK, zwar gibt es das Menu noch, aber sämtliche Menu-Punkte werden funktionslos.

&H36E Sofern das BASIC-Window aktiv ist, werden sämtliche Tastendrücke ignoriert.

2.5.4.3 Programmgesteuertes WINDOW

Mit der Maus läßt sich (fast) jedes Window beliebig auf dem Bildschirm positionieren. Soweit ist das ja auch recht schön, aber nicht immer möchte man dazu die Maus benutzen. Programmgesteuert ist das doch viel eleganter. Die "intuition.library" macht's wieder möglich:

```
LIBRARY "intuition.library"

WINDOW 2,"Hello!",(10,10)-(350,150),0
WINDOW OUTPUT 2

PRINT "Hello!"

FOR t=1 TO 5000:NEXT t

CALL MoveWindow(WINDOW(7),20,30)
FOR t=1 TO 2000:NEXT t
PRINT "Ich habe mich gerade bewegt!"

CALL MoveWindow(WINDOW(7),-20,-30)
FOR t=1 TO 2000:NEXT t
PRINT "Schon wieder!"
FOR t=1 TO 10000:NEXT t

WINDOW CLOSE 2

LIBRARY CLOSE
END
```

Achtung: Der Intuition-Befehl MoveWindow kontrolliert Ihre Eingaben nicht auf Fehler. Falls Ihre Delta-Werte in den Ecken eines fernen Universums liegen, dann versucht Intuition, das Window in die Ecken eines fernen Universums zu bewegen. Auf Grund der Störungen im Raum-Zeit-Kontinuum, die daraus resultieren können, ist das Ergebnis kein schöner Anblick.

Bei den Delta-Werten handelt es sich um die Verschiebung in x- bzw. y-Richtung von der linken oberen Ecke des Windows aus gesehen. Sollte sich auf Grund Ihrer Verschiebung auch nur ein klitzekleiner Teil des Windows aus dem Bildschirmbereich davonstehlen, so kontert der Amiga mit einer Guru Meditation.

2.5.4.4 Move Screen

Was für das Window gilt, hat natürlich auch für den Screen seine Gültigkeit! Auch der Screen kann programmgesteuert hoch- und runtergescrollt werden. Diesmal hilft der Intuition-Befehl:

```
MoveScreen(Screen,DeltaX,DeltaY)
```

Wie auch bei MoveWindow handelt es sich um Delta-Verschiebungen, die relativ zur gegenwärtigen Position sind. Screen ist ein Zeiger auf die Screen-Struktur des gewünschten Screens. Wir können den Zeiger leicht aus der bereits bekannten WINDOW-Struktur auslesen:

```
screen=PEEKL(WINDOW(7)+46)
```

So läßt sich der Screen verschieben:

```
LIBRARY "intuition.library"

FOR x=1 TO 280
ScreenDown
NEXT x

FOR x=1 TO 280
ScreenUp
NEXT x

LIBRARY CLOSE
END


SUB ScreenUp STATIC

sc&=PEEKL(WINDOW(7)+46)
CALL MoveScreen(sc&,0,-1)

END SUB

SUB ScreenDown STATIC

sc&=PEEKL(WINDOW(7)+46)
CALL MoveScreen(sc&,0,1)

END SUB
```

Es ist jedoch derzeitig unmöglich, einen Screen in x-Richtung zu verschieben.

2.5.4.5 SetWindowTitles - Windows benennen

Ist Ihnen aufgefallen, daß jedes WINDOW einen eigenen Namen besitzt? Da gibt es das BASIC-Window (bzw. "der-Name-Ihres-Programmes"-Window), ein LIST-Window, und jedes neuerschaffene Window wird ebenfalls mit einem wohlklingenden Namen versehen. Als Spielerei ist das wunderschön, aber diese Namen lassen sich auch programmtechnisch sehr vielseitig nutzen. So könnte man beispielsweise das Ausgabewindow mit dem Namen des gerade aktivierten Menü-Punktes belegen, eine Copyright-Meldung oder einen wichtigen Benutzerhinweis einfügen, etc., etc. Aber wie lassen sich die Window-Namen nun verändern, ohne gleich ganz neue Windows zu schaffen?

Mit der folgenden SUB-Routine kein Problem! Der Aufruf erfolgt durch:

```
SetTitle window$,screen$
```

window$ ist dabei der neue Name des Ausgabewindows (kann durch WINDOW OUTPUT bestimmt werden). Mit screen$ kann sogar der Name des Screens verändert werden, in dem sich das Window befindet. Um einen Namen unverändert zu lassen, benutzt man als Namen CHR$(0).

```
LIBRARY "intuition.library"

SetTitle "(C) 1986 by Tobe", "Amiga Tips&Tricks"

LIBRARY CLOSE
END
```

```
SUB SetTitle(wind$,scr$) STATIC

wind$=wind$+CHR$(0)
scr$=scr$+CHR$(0)
CALL SetWindowTitles(WINDOW(7),SADD(wind$),SADD(scr$))

END SUB
```

Der neue Screen-Title wird natürlich nur angezeigt, wenn auch das aktuelle BASIC-Window aktiv ist.

2.5.4.6 Window-Limits

Spielen Sie einmal ein bißchen mit dem Sizing-Gandet Ihres BASIC-Windows herum! Sicherlich werden Sie merken, daß sich Windows nur bis zu einer bestimmten Größe verkleinern können. In der Intuition-Windowstruktur gibt es vier Speicherwerte, die die kleinst- und größtmögliche Ausdehnung eines Windows festlegen.

Mit der Intuition-Funktion WindowLimits lassen sich diese Limits nach Herzenslust verändern. Die folgende Routine erledigt diese Aufgabe und wird durch:

```
Limit minX, minY, maxX, maxY
```

aufgerufen.

Achtung: Wenn Ihre Minimum-Werte größer sind als das betreffende Window auf dem Schirm, oder wenn Ihre Maximal-Werte kleiner sind als das Window, kommt es zu einer Fehlermeldung.

```
'Mit diesem Programm koennen Sie Ihr
'BASIC-Window der neuen Workbench (Ver-
'sion 1.2) anpassen; es laesst sich
'nun auf volle Bildschirmgroesse ver-
'groessern (640*270)


DECLARE FUNCTION WindowLimits& LIBRARY

LIBRARY "intuition.library"

Limit 5,5,640,270

LIBRARY CLOSE
END

'-------------------------------

SUB Limit(minX%,minY%,maxX%,maxY%) STATIC

e&=WindowLimits&(WINDOW(7),minX%,minY%,maxX%,maxY%)
IF e&=0 THEN
  PRINT "Min.-Werte groesser als Window oder"
  PRINT "Max.-Werte kleiner als Window"
  ERROR 255
END IF
```

2.6 Memory-Handling

Das gesamte Betriebssystem des Amiga war von vornherein darauf ausgelegt worden, Erweiterungen und Verbesserungen gegenüber so offen wie möglich zu stehen. Hinzu kamen die Anforderungen, die man an einen Multitasking-Computer stellen muß, und das Ergebnis war eine besondere Art der Speicherverwaltung: Sieht man von Adresse 4 und den Hardware-Adressen ab, dann gibt es keine absoluten Speicherplatz-Belegungen.

Wer zuerst kommt, mahlt zuerst. Programme, die in C oder Assembler geschrieben wurden, können frei im Speicher verschoben werden. Auch der AmigaBASIC Interpreter kann zu verschiedenen Zeitpunkten an ganz verschiedenen Stellen im Speicher liegen.

Es gibt allerdings einen grundsätzlichen Speicheraufbau, den wir Ihnen natürlich nicht vorenthalten wollen:

Adreß-Bereich-Belegung:

000000-3FFFFF	RAM der ersten 256K
000000-07FFFF	RAM mit 256K-Speichererweiterung
200000-9FFFFF	8 MByte für Erweiterungen
A00000-BFFFFF	Platz für externe Decoder Erweiterungen
BFD000-BFDF00	8520-B (nur an geraden Adressen adressiert)
BFE001-BFEF01	8520-A (nur an ungeraden Adressen adressiert)
C00000-DFEFFF	Reserviert für zukünftige Erweiterungen
DFF000-DFFFFF	Spezial Chips
E00000-E7FFFF	Reserviert für zukünftige Erweiterungen
E80000-EFFFFF	Expansion Slot Decoding
F00000-F7FFFF	Reserviert für zukünftige Erweiterungen
F80000-FFFFFF	SYSTEM ROM oder Kickstart WOM

2.6.1 Speicher durch Variablen

Einen einfachen Weg, RAM-Speicher zu reservieren, ist die Verwendung eines Variablenfeldes. Dazu müssen Sie lediglich ein entsprechend großes Variablenfeld dimensionieren. Anschließend erhalten Sie die Anfangsadresse im Speicher durch den AmigaBASIC-Befehl VARPTR:

```
DIM picture%(100)
add&=VARPTR(picture%(0))
```

Mit diesem Beispielprogramm stünden Ihnen beispielsweise 200 Bytes (Integer=2 Bytes) ab add& im Speicher zur Verfügung.

2.6.2 Speicher reservieren

Zwei Befehle aus Intuition bzw. Exec, liefern Ihnen soviel Speicherplatz wie Sie brauchen (vorausgesetzt, Ihr Amiga spielt mit). Es handelt sich dabei um die Exec-Funktion AllocMem und die Intuition-Funktion AllocRemember.

Sie können frei wählen, aus welchem der folgenden Speicherblöcke Sie gern ein Stück hätten; eine konkrete Anfangsadresse können Sie jedoch niemals im voraus festlegen.

PUBLIC	2^0
CHIP	2^1 Für DMA, etc.
FAST	2^2 Non-Chip-Speicher
CLEAR	2^16 Löscht den Speicher vor Übergabe

AllocMem

AllocMem reserviert einen beliebig langen Speicherblock für Sie. Es liegt in Ihrer Verantwortung, diesen nach Gebrauch via FreeMem zurückzugeben:

```
DECLARE FUNCTION AllocMem& LIBRARY
LIBRARY "exec.library"

opt&=2^1+2^16
buffer&=AllocMem&(1000,opt&)
IF buffer&=0 THEN ERROR 7

(...)

CALL FreeMem(buffer&,1000)
LIBRARY CLOSE
END
```

AllocRemember

AllocRemeber hilft Ihnen, wie der Name schon sagt, beim Erinnern. Sie können AllocRemember beliebig oft aufrufen, um ebenso viele Speicherblöcke zu erhalten. Brauchen Sie diese nicht mehr, dann können alle Blöcke mit einem Schlag durch FreeRemember freigegeben werden, ohne daß Sie sich noch um Größe und Anfangsadressen der einzelnen Blöcke kümmern müssen:

```
DECLARE FUNCTION AllocRemember& LIBRARY
LIBRARY "intuition.library"

opt&=2^1+2^16  'CHIP und CLEAR
buffer1&=AllocRemember&(0,1000,opt&)
IF buffer&1=0 THEN ERROR 7

(...)

buffer2&=AllocRemember&(0,7650,opt&)
IF buffer2&=0 THEN ERROR 7

(...)

CALL FreeRemember(0,-1)
LIBRARY CLOSE
END
```

Diese Methode benötigt ein wenig mehr Speicherplatz als AllocMem, da zum "Merken" der Parameter eigens eine kleine Datenstruktur angelegt werden muß.

2.7 Das Disketten-Laufwerk

Das interne 3.5 Inch Disk Drive des Amiga besitzt zwei Schreib/Lese-Köpfe, die gleichzeitigen Zugriff auf beide Diskettenseiten zulassen. Die Diskette selbst wird pro Seite in 80

Zylinder unterteilt. Jeder Zylinder besteht aus 11 Sektoren, die wiederum jeweils 512 benutzbare Datenbytes sowie 16 Sektorverwaltungsbytes beinhalten. Damit stehen insgesamt:

2 Köpfe *
80 Zylinder *
11 Sektoren *
512 Bytes =

und 901120 Bytes (880 KByte) Data sowie 28160 Bytes (28 KByte) Label zur Verfügung.

Der Amiga Disk Controller kann es mit bis zu vier 3.5-, bzw. 5.25-Inch-Drives aufnehmen. Durch DMA (Direct Memory Access) können diese Drives weitgehend unabhängig vom übrigen System betrieben werden.

Um an die Disk Drives heranzukommen, kann man sich verschiedener Anwendungsebenen bedienen:

1. Programm
2. Command Line Interface
3. AmigaDOS
4. Trackdisk Device
5. Hardware direkt

Gleich vorweg sei gesagt, daß dem AmigaBASIC-Programmierer alle fünf Möglichkeiten offenstehen.

2.7.1 Das CLI in AmigaBASIC-Programmen

Das Command Line Interface (CLI), jene in der "System"-Schublade verborgene Schnittstelle Mensch - DOS, kann auch in AmigaBASIC-Programmen direkt verwendet werden! Die etwas spärlich ausgefallenen AmigaBASIC-eigenen Diskettenbefehle werden nun von einer wahren Fülle neuer Befehle ergänzt, die sich alle um das Thema Disk drehen.

Das folgende Programm verwendet die DOS-Library und stellt den neuen Befehl "CLI" zur Verfügung, mit dem alle CLI-Befehle ausgeführt werden können. Das Format ist:

```
CLI "command-string"
```

zum Beispiel:

```
CLI "list df0:fonts keys to prt:"
```

würde das Unter-Directory der Disk in Laufwerk 0 (intern) sowie die Blocknummern (Keys) der einzelnen Files auf den Drucker ausgeben.

Programm-Größe: 2291 Bytes
Bemerkungen: "dos.bmap" muß sich auf Disk befinden

```
'CLI

'*****************************************
'*                                       *
'* (C) 1986 by DATA BECKER GmbH W.-Ger.  *
'* (P) by Tobias "Kutte" Weltner         *
'*                                       *
'* Dieses Programm  ermoeglicht den Ein- *
'* satz des Command Line Interface (CLI) *
'* durch  AmigaBASIC  Programme. Weitere *
'* Informationen  und  Programme  hierzu *
'* finden Sie  in  DATA BECKERs  "AMIGA *
'* Tips & Tricks".                       *
'*                                       *
'*****************************************
```

```
DECLARE FUNCTION DosOpen& LIBRARY      ´Eroeffnen des CLI-Win
  dows
DECLARE FUNCTION DosExecute& LIBRARY   ´Ausfuehren eines CLI-
  Befehls

LIBRARY "dos.library" ´alles zu Finden in der DOS-Library

main:´===============(demo)====================
                                                    ´=
CLI "list df0: KEYS"                                ´=
                                                    ´=
LIBRARY CLOSE                                       ´=
END                                                 ´=
                                                    ´=
´==============================================


SUB CLI(execute$) STATIC

´---------------------------------------------
´Befehlsformat: CLI "CLI-Befehl/Befehls-Sequenz"  -
´---------------------------------------------

command$=execute$+CHR$(0)              ´Befehl mit ´0´ abschl
  iessen
wind$="CON:10/10/600/190/CLI"+CHR$(0)  ´CLI-Window definieren


f&=DosOpen&(SADD(wind$),1006)          ´Window eroeffnen
IF f&=0 THEN                           ´will nicht
  ERROR 255                            ´ERROR ausgeben
END IF

s&=DosExecute&(SADD(command$),0,f&)    ´CLI-Befehl ausfuehren
```

```
                                   'command$=CLI-Befehl
IF s&=0 THEN                       'Befehl konnte nicht a
  usge-
  ERROR 255                        'fuehrt werden (s&=0)
END IF

FOR t=1 TO 10000:NEXT t

CALL DosClose(f&)                  'CLI-Window wieder sch
  liessen

END SUB                            'das war's
```

Variablen:

execute$:	die Befehlssequenz
command$:	0-terminated
wind$:	CLI-Window Parameter für Console Device
f&:	File Handle für CLI-Window
s&:	Fehler-Flag

Programmbeschreibung:

Die beiden DOS-Funktionen Open und Execute werden als Funktionen deklariert. Nach dem Öffnen der benötigten Library wird der CLI-Befehl einer Probe unterzogen.

Der Befehlstext in execute$ wird nach command$ transferiert, wo er mit einem Null-Byte abgeschlossen wird. Über das Console Device wird in den nächsten beiden Zeilen ein Window geöffnet. Dies ist erforderlich, weil der Execute-Befehl, der schließlich den CLI-Befehl ausführt, eine AmigaDOS-File Handle als Ausgabe-Adresse erwartet.

DosExecute wird die Anfangsadresse des Befehls-Strings sowie Input- und Output-Parameter geliefert. Kein weiterer Input wird erwartet (=0), die Ausgabe erfolgt nach f&, dem neuen Window.

Falls der gewünschte CLI-Befehl nicht wunschgemäß ausgeführt werden kann, meldet Execute in s& einen Fehler (s&=0). Andernfalls wird auf einen Maus-Click gewartet, das Window wieder geschlossen und CLI verlassen.

2.7.2 Programm-Kommentare setzen

Jedes Programm oder Directory kann mit einem bis zu 80 Zeichen langen Kommentar versehen werden. Oft gibt der Programm-Name allein nicht genügend Auskunft über Sinn und Art einer Anwendung. Kommentare wie "noch in der Entwicklung!", "geschrieben von Vogi" oder "Version 3.4" beinhalten wichtige Informationen über das betreffende Programm. Auch Directory-Kommentare wie "In diesem Directory liegen die .bmap-Files" oder "Hier sind die Geschäftsbriefe abgelegt" sind nicht minder wirkungsvoll.

Das folgende Programm kann ein beliebiges File oder Directory mit dem gewünschten Kommentar versehen. Werden diese Files dann später beispielsweise mit dem CLI-Befehl LIST untersucht, so taucht der Kommentar wieder auf.

Das Format des Befehls ist:

```
SetComment "programm-name", "kommentar"
```

Beispielsweise versieht...

```
SetComment "df0:Clock", "Dies ist eine Uhr!!!"
```

das File "Clock" mit einem Kommentar.

Programm-Größe: 1045 Bytes
Bemerkungen: "dos.bmap" muß sich auf Disk befinden

```
'SetComment

DECLARE FUNCTION DosSetComment& LIBRARY

LIBRARY "dos.library"

main:
SetComment "compare", "Ein absolut sinnloses Programm...!"
'kommentiert ein Programm namens "compare"

LIBRARY CLOSE
END

'---------------------------------------

SUB SetComment(file$,comment$) STATIC

'Ein Kommentar von max. 80 Zeichen kann an jedes DOS-File
'angehaengt werden. Der Kommentar wird beispielsweise von
'der CLI-Funktion "list" wieder ausgegeben.

SHARED fl                        'fl=1:I/O-Error
file$=file$+CHR$(0)              'Namen mit '0'
comment$=comment$+CHR$(0)        'abschliessen

suc&=DosSetComment&(SADD(file$),SADD(comment$))
IF suc&=0 THEN
  PRINT "Schon wieder ein Problem...!"
  PRINT "z.B. Programm gibt's nicht oder"
  PRINT "Diskette write-protected"
  fl=1
END IF

END SUB
```

Variablen:

file$:	Name des gewünschten Files oder Directories
comment$:	Kommentar
fl:	I/O-Error Flag
suc&:	Fehler-Flag

Programmbeschreibung:

Die nötige Funktion DosSetComment wird deklariert und die DOS-Library geöffnet. Dem SUB-Programm werden File-Name (file$) und Kommentar (comment$) übergeben. Die Fehlervariable fl wird allgemein zugänglich gemacht (sie soll ja später ans Hauptprogramm zurückgeliefert werden). Die beiden Text-Strings werden mit einem Null-Byte versehen, und die Funktion DosSetComment wird aufgerufen. Im Normalfall geht alles gut, und der Kommentar wird ordnungsgemäß an das ahnungslose Programm gehängt. Geht jedoch etwas schief (das File gibt's gar nicht oder es ist schreibgeschützt), dann ist suc&=0. Eine Problemmeldung wird ausgegeben und das Fehlerflag fl gesetzt.

2.7.3 CheckFile - wirklich auf Disk?

Ist ein Datenfile nun auf Disk oder nicht? Oftmals ist diese Frage außerordentlich wichtig, denn nur ein existierendes File kann auch geöffnet werden; andernfalls kommt es zu einer Fehlermeldung.

Das folgende Programm kann Ihnen helfen. Der neue Befehl:

```
CheckFile "filename"
```

untersucht, ob es das angegebene File gibt. Aber die Routine kann noch mehr! Gleichzeitig wird die Nummer des Anfangs-

blocks auf der Diskette ausgegeben. Dadurch können Sie (mit einem DiskEditor beispielsweise, siehe trackdisk.device) auf das gewünschte Programm direkt zurückgreifen und es auf der Disk Block für Block untersuchen.

Programm-Größe: 857 Bytes
Bemerkungen: "dos.bmap" muß sich auf der Disk befinden

```
'CheckFile

DECLARE FUNCTION DosLock& LIBRARY

LIBRARY "dos.library"

main:

LINE INPUT "Gesuchtes File --> ";fl$
CheckFile fl$

IF fl=1 THEN
  PRINT "File existiert."
  PRINT "File Header beginnt ab Block";blk&;" auf Disk."
ELSE
  PRINT "Das File gibt es nicht!"
END IF

LIBRARY CLOSE
END

'--------------------------------------

SUB CheckFile (file$) STATIC

SHARED fl,blk&                      'fl=0: File existiert nicht
                                    'fl=1: File existiert
                                    'blk&: # des Blockes auf Disk

file$=file$+CHR$(0)                 'Namen mit '0' abschliessen
accessRead%=-2                      'SHARED_ACCESS
```

```
lock&=DosLock&(SADD(file$),accessRead%)
IF lock&=0 THEN                         'File gibt's nicht
  fl=0
ELSE                                    'File gefunden
  fl=1
  blk&=PEEKL(lock&*4+4)
END IF

CALL DosUnLock(lock&)

END SUB
```

Variablen:

fl:	fl=0: File existiert nicht
blk&:	Block-Nummer des File-Headers auf Disk
file$:	Name des gewünschten Files
accessRead%:	SHARED ACCESS (-2)
lock&:	Adresse auf File Lock des Files

Programmbeschreibung:

Die DOS-Funktion Lock ist eine Art Saugfuß. Mit ihrer Hilfe kann sich nämlich DOS an einem ganz bestimmten File oder Directory "festsaugen". Eine Lock-Datastruktur wird eingerichtet, die die speziellen Parameter dieses Files (oder Directories) enthält.

Das Programm verwendet nun DosLock, um zu überprüfen, ob ein bestimmtes File existiert. Hierzu wird der Name des Files (file$) mit einem Null-Byte versehen (Ende-Kennzeichen), der SHARED ACCESS Mode gewählt (näheres dazu gleich) und ein Lock für das File angefordert. Gibt es das File nicht, so kann natürlich auch keine Lock-Datastruktur angelegt werden, und lock& enthält 0. Andernfalls enthält lock& die Anfangsadresse der Datastruktur. Dabei handelt es sich um einen sogenannten BPTR, der nicht den absoluten Adressenwert enthält, sondern

die Longword-Adresse. Da jedes Longword jedoch genau 4 Bytes enthält, ergibt sich der Adresspointer durch einfaches Multiplizieren (*4).

Genau dies geschieht in den folgenden Anweisungen, wo die Blocknummer des Anfangsblocks aus der Datenstruktur gelesen wird.

Schließlich muß das System-Lock wieder entfernt werden, denn man wollte ja nur mal gucken, ob das File auch wirklich da ist. Dies erledigt DosUnLock für Sie.

Anfangs wurde erwähnt, daß das Programm den sogenannten SHARED ACCESS Mode verwendet. Im Grunde kann der Saugfuß nämlich in zwei Weisen benutzt werden:

1. SHARED ACCESS (-2): Dieser Mode wird auch ACCESS READ genannt. Mit ihm erhalten Sie die Datenstruktur des Files, ohne es für andere Benutzer (andere laufende Tasks) zu blockieren. Während Sie also dieses File untersuchen, könnte ein anderer Task dasselbe File ebenfalls untersuchen. Dies geht auch problemlos, da Sie das File ja lediglich lesen.

2. EXCLUSIVE WRITE (-1): Mit diesem Mode können Sie ein File ganz für Sie allein pachten. Solange Sie das System Lock nicht durch DosUnLock wieder freigeben, sind Sie der einzige Task, der mit dem File arbeiten kann. Andere Tasks werden mit der Meldung "FILE ALREADY OPEN" (File bereits geöffnet) abgespeist. Sie sollten diesen Mode allerdings mit Vorsicht genießen, denn schließlich würde es Sie ja auch stören, wenn ein anderer Task sich ein File "geklaut" hat, auf das Sie auch scharf waren. Dieser Mode hat allerdings gewisse Berechtigung, wenn Sie ein File beschreiben wollen, den Inhalt des Files also ändern. In diesem Fall wäre es nämlich hinderlich, wenn ein anderes Programm dasselbe Programm mit anderen Daten vollstopft, während Sie Ihre eigenen Änderungen vornehmen. Selbst wenn ein anderer

Task das File "nur" liest, während Sie Ihre Adresskartei in diesem File ändern, könnte es zu gewissen Mißverständnissen kommen...!

Hier nun der Vollständigkeit halber die erwähnte Lock-Datastruktur:

```
00 - 03  L  BPTR(!) zum nächsten Lock, sonst 0
04 - 07  L  Blocknummer des Dir's oder Fileheaders
08 - 11  L  Shared (-2) oder Exclusive (-1) Access
12 - 15  L  APTR Process ID des Handlertasks
16 - 19  L  BPTR(!) zum Disk-Eintrag
```

Weitere Informationen über Process ID und Diskeinträge finden Sie an anderer Stelle in diesem Buch.

Anmerkung: Mit diesem Programm können Sie beispielsweise prüfen, ob sich die entsprechenden ".bmap"-Files auf Disk befinden, bevor Sie Kernel-Befehlsroutinen verwenden.

2.7.4 Daten sichern: DOS-Protection

Vielleicht haben Sie schon einmal die grauenhafte Erfahrung gemacht: Freudig haben Sie ein ganz wahnsinnig tolles Programm geschrieben, und nun soll es auf Diskette abgespeichert werden. Am besten unter dem Namen "test". Gesagt, getan, doch zu spät! Zu spät ist Ihnen eingefallen, daß Sie gestern ein noch viel tolleres Programm unter demselben Namen abgespeichert hatten, und sang- und klanglos wird Ihr altes Programm vom neuen "überspielt".

In der bisherigen Version von AmigaBASIC wird nämlich ohne Vorwarnung munter drauflos gesaved, ohne vorher nachgeprüft zu haben, ob es ein gleichnamiges File bereits gibt. Diesem Dilemma kann jedoch Abhilfe geschaffen werden, denn Sie können auf recht simple Weise Programme "überschreibungssicher" machen. In jedem Programm-Header gibt es nämlich vier Bits, die die folgenden Bedeutungen haben:

Bit 1: DELETE
Bit 2: EXECUTE
Bit 3: WRITE
Bit 4: READ

Sie können also Ihr Programm gegen Überschreiben (DELETE), unbefugtes Starten (EXECUTE), Verändern (WRITE) und Lesen (READ) schützen! Theoretisch. In der bisherigen DOS-Version wird nämlich lediglich Bit 1 überprüft; sie können Programme also zunächst nur lösch-sicher machen.

Wie dem auch sei, das folgende Programm beschert Ihnen den Protect-Befehl, mit dem Sie ein jegliches Programm mit einem oder mehreren der folgenden Attribute ausstatten können. Um Bits und Bytes brauchen Sie sich nicht zu kümmern, das Befehlsformat sieht so aus:

```
Protect "filename","read|write|execute|delete"
```

(Die |-Taste befindet sich übrigens in der rechten oberen Ecke Ihrer Tastatur...)

Sie können die vier Schutzmodi in beliebiger Reihenfolge angeben. Das |-Zeichen muß zwischen jedem Wort stehen.

Programm-Größe: 1293 Bytes
Bemerkungen: "dos.bmap" muß sich auf Disk befinden

```
'Protect

DECLARE FUNCTION DosSetProtection& LIBRARY
LIBRARY "dos.library"

Protect "tt","read¦write¦execute¦delete"

LIBRARY CLOSE
END
```

```
SUB Protect(file$,mask$) STATIC

subPVar:                              'Variablen
SHARED fl                             'fl=1: I/O-Error
file$=file$+CHR$(0)                   'Name mit '0' abschliessen
prot$(3)="READ"                       'Leseschutz
prot$(2)="WRITE"                      'Schreibschutz
prot$(1)="EXECUTE"                    'Start-Schutz
prot$(0)="DELETE"                     'Loesch-Schutz

FOR loop%=1 TO LEN(mask$)             'Attribute selektieren
  byte$=MID$(mask$,loop%,1)
  IF byte$<>CHR$(124) THEN            'Zeichen="|"?
    p$(count%)=p$(count%)+byte$
  ELSE                                'ja
    count%=count%+1
  END IF
NEXT loop%
FOR loop%=3 TO 0 STEP -1              'Bit-Mask setzen
  FOR loop2%=0 TO 3
    IF UCASE$(p$(loop2%))=prot$(loop%) THEN
      mask%=mask%+2^loop%
    END IF
  NEXT loop2%
NEXT loop%

'READ    = 2^3 = 8
'WRITE   = 2^2 = 4

'EXECUTE = 2^1 = 2
'DELETE  = 2^0 = 1

suc&=DosSetProtection&(SADD(file$),mask%)
IF suc&=0 THEN
  PRINT "Wir haben ein Problem..." 'I/O-Error
  fl=1
END IF

END SUB
```

Variablenfelder:

prot$: Schutz-Modi
p$: Wort-Speicher

Variablen:

fl: Fehlerflag
file$: Programmname, mit 0-Byte versehen
mask$: Modi-Maske
mask%: Bit-Maske
byte$: Ein-Byte-Zeichen aus Maske
count%: Zeichenzähler
loop%: Schleifenvariable
loop2%: noch eine Schleifenvariable
suc&: Fehlerflag der DosProtect-Funktion

Programmbeschreibung:

Die Variable fl dient als Fehlerflag. fl=1 steht für einen I/O-Fehler. Das gewünschte Programm konnte also nicht geschützt werden, weil es z.B. nicht existierte oder nicht im aktuellen Directory zu finden war.

Der Programmname in file$ muß mit einem 0-Byte versehen werden, und die drei Schutzmodi werden als Name definiert. Die hier abgespeicherten englischen Namen werden später mit den Namen beim Funktionsaufruf verglichen. Sind Ihnen also deutsche (oder andere) Bezeichnungen lieber, so können Sie hier die gewünschten Bezeichnungen eintragen.

Eine Schleife geht nun mask$ Zeichen für Zeichen durch. Erst wenn ein |-Zeichen gefunden wird, gilt das eingelesene Wort als beendet. Die gefundenen Worte werden in p$(x) abgespeichert.

Eine zweite Schleife untersucht die gefundenen Worte. Es handelt sich dabei praktisch um zwei ineinander verschachtelte Schleifen, die jedes Wort in jeder beliebigen Reihenfolge mit den in prot$(x) vorgegebenen Schlüsselwörtern vergleicht. Dadurch können Sie bei Funktionsaufruf die Schutz-Parameter in ganz beliebiger Reihenfolge angeben. Entsprechend den Funden werden die oben näher beschriebenen Protection-Bits gesetzt und in mask% abgespeichert.

DosSetProtection versieht dann das Programm mit den Bits. Gibt es dabei Probleme, wird eine Warnung ausgegeben und die Fehlervariable fl gesetzt.

Den Schutz-Zustand eines jeden Files können Sie sich beispielsweise mit dem CLI-Befehl "LIST" anschauen. Im Normalfall finden Sie dort die Bezeichnung "rwed" für read write execute delete. Das bedeutet, das File ist ungeschützt. Schützen Sie es beispielsweise gegen überschreiben, wäre das Ergebnis "rwe-", das File ist nicht mehr "deletable".

```
PROTECT "workbench:Clock","delete"
```

Im CLI-List-Ausdruck fänden Sie den Eintrag "rwe-".

Wollen Sie einen Programmschutz wieder rückgängig machen, so brauchen Sie bloß das entsprechende File neu zu schützen und den unerwünschten Schutz wegzulassen:

```
PROTECT "workbench:Clock",""
```

ließe Clock völlig ungeschützt zurück.

2.7.5 Rename

Nun noch schnell ein DOS-Rename-Programm, das den Namen eines beliebigen Files (oder Directories) wunschgemäß verändert. Der Aufruf dieser Funktion geschieht durch:

```
RENAME "alter Name", "neuer Name"
```

Programm-Größe: 519 Bytes
Bemerkungen: "dos.bmap" muß sich auf Disk befinden

```
'Rename

DECLARE FUNCTION DosRename& LIBRARY
LIBRARY "dos.library"

Rename "hi","s-z"

LIBRARY CLOSE
END

'-----------------------------------

SUB Rename(oldName$,newName$) STATIC

SHARED fl                             'fl=1: I/O-Error
oldName$=oldName$+CHR$(0)             'jeweiligen Namen
newName$=newName$+CHR$(0)             'mit '0' abschliessen

suc&=DosRename&(SADD(oldName$),SADD(newName$))

IF suc&<>-1 THEN
  PRINT "Wir haben ein Problem..."
  fl=1
END IF

END SUB
```

Variablen:

fl:	Fehlerflag
oldName$:	alter Name
newName$:	neuer Name
suc&:	Fehlerflag der DOS-Funktion

Programmbeschreibung:

Sowohl der alte als auch der neue Name müssen mit einem 0-Byte abgeschlossen werden. Die beiden Adressen der Namen werden der DOS-Funktion DosRename übergeben, welche die Namensänderung vornimmt. Geht etwas schief, so wird eine Warnung ausgegeben und das Fehlerflag gesetzt.

2.7.6 GetDir - Einblick ins Inhaltsverzeichnis

Es kommt oft vor - ein Programm benötig dringend Einblick in das Disketten-Inhaltsverzeichnis einer Diskette. Der Befehl FILES hilft da nur bedingt, denn ohne Differenzierung gibt er die Namen der entsprechenden Files gleich auf den Bildschirm aus. Deshalb haben wir die Routine "GetDir" geschrieben, die alle wichtigen Daten aus dem aktuellen Directory liest und in Variablenfeldern abspeichert. Was mit den Daten dann geschieht, ist ganz und gar Ihre Sache!

Programm-Größe: 4101 Bytes
Bemerkungen: "exec.bmap" und "dos.bmap" müssen sich auf Disk befinden

```
'GetDir

DECLARE FUNCTION DosExamine& LIBRARY
DECLARE FUNCTION DosExNext& LIBRARY
DECLARE FUNCTION DosLock& LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION DosIoErr& LIBRARY

LIBRARY "exec.library"
LIBRARY "dos.library"

Arrays:                    'Felder

x%=100                     '100 Eintraege (max.)
DIM SHARED DirName$(x%)    'File-Name
DIM SHARED DirProt$(x%)    'Protection-Bits
DIM SHARED DirType$(x%)    'Type: DIR oder FILE
DIM SHARED DirSize&(x%)    'FILE-Groesse in Bytes...
DIM SHARED DirBlks&(x%)    '...und Blocks auf Disk
DIM SHARED DirComm$(x%)    'Kommentar
```

```
main:

GetDir "df0:",x%
GOSUB PrintDirectory

LIBRARY CLOSE
END

'------------------------------------------------

PrintDirectory:

FOR loop%=0 TO counter%
  PRINT "Nr. ";loop%+1
  PRINT DirName$(loop%)
  PRINT "Protection:  ";DirProt$(loop%)
  PRINT "Type:        ";DirType$(loop%)
  PRINT "Size:        ";DirSize&(loop%)
  PRINT "Bloecke:     ";DirBlks&(loop%)
  PRINT "Kommentar:   ";DirComm$(loop%)
  PRINT STRING$(50,"-")
  PRINT STRING$(50,"-")
  SLEEP:SLEEP
NEXT loop%

RETURN

'-------------------------------------------

SUB GetDir(dir$,max%) STATIC

SubGDVar:                  'Variablen
SHARED counter%,fl         'counter%: # der Eintraege in die
  sem DIR
                           'fl=1: Directory existiert nicht
                           'fl=2: Kein freier Speicherplatz
                           'fl=3: I/O Error
AccessRead%=-2             'allgemeiner Zugriff (SHARED)
dir$=dir$+CHR$(0)          'DIR-Name mit '0' abschliessen
bytes&=252                 'Buffer-Groesse

PRINT "* searching"
PRINT

GetDir:
lock&=DosLock&(SADD(dir$),AccessRead%)
IF lock&=0 THEN
  PRINT "Directory existiert nicht."
  fl=1
END IF
```

```
AllocateMemory:

opt&=2^1+2^16                               'MEMF_CLEAR|MEMF_CHIP
info&=AllocMem&(bytes&,opt&)
IF info&=0 THEN                             'Out of Memory
  PRINT "Kein Speicherplatz frei!"
  fl=2
  GOTO ende

END IF

StartExamination:

suc&=DosExamine&(lock&,info&)
IF suc&=0 THEN                              'I/O-Error
  PRINT "Directory konnte nicht untersucht werden."
  fl=3
  GOTO finish
END IF

again:

DirName&=info&+8                            'Adresse des DIR-Namens
  in info&

FOR search=0 TO 29
  check=PEEK(DirName&+search)               'schon Ende (=0) ?
  IF check<>0 THEN                          'nein
    check$=check$+CHR$(check)
  ELSE                                      'ja
    search=29                               'Schleife vorzeitig verl
  assen
  END IF
NEXT search

DirName$(counter%)=check$:check$=""  'Name in Feld ablegen
prot&=PEEKL(info&+116)                      'Protection-Bits untersu
  chen

IF prot&<>0 THEN                            'welche da?
  IF (prot& AND 2^3)<>0 THEN prot$=prot$+"read-,"
  IF (prot& AND 2^2)<>0 THEN prot$=prot$+"write-,"
  IF (prot& AND 2^1)<>0 THEN prot$=prot$+"execute-,"
  IF (prot& AND 2^0)<>0 THEN prot$=prot$+"delete-,"
  DirProt$(counter%)=LEFT$(prot$,LEN(prot$)-1)+"protected."
  prot$=""
END IF

type&=PEEKL(info&+120)                       'Eintags-Type untersuch
```

```
en
IF type&<0 THEN                               'FILE'
  DirType$(counter%)="FILE"
ELSEIF counter%=0 THEN                        'unser DIR
  DirType$(counter%)="CURR.DIR"
ELSE
  DirType$(counter%)="DIR"                    'DIR
END IF

DirSize&(counter%)=PEEKL(info&+124)  'FILE Byte-Groesse
DirBlks&(counter%)=PEEKL(info&+128)  'FILE Blocks on Disk

FOR search=0 TO 79                            'Kommentar suchen
  check=PEEK(info&+144+search)
  IF check<>0 THEN
    check$=check$+CHR$(check)
  ELSE
    search=79
  END IF
NEXT search

DirComm$(counter%)=check$:check$=""  'Kommentar speichern

suc&=DosExNext&(lock&,info&)                  'ein weiterer Eintrag?
IF suc&=0 THEN                                'nein
  e&=DosIoErr&
  IF e&<>232 THEN                             '232=legal error NO_MOR
  E_ENTRIES
    PRINT "Fehler im Directory. I/O-Error Nr. ";e&
    DirError&=e&
  END IF
  GOTO finish
ELSE                                          'ja
  counter%=counter%+1                         'naechster Eintrag
  IF counter%<=max% THEN
    GOTO again
  END IF
END IF

finish:
```

```
CALL FreeMem(info&,bytes&)
CALL DosUnLock(lock&)

ende:

END SUB
```

Variablen-Felder:

DirName$:	Name des Eintrages
DirProt$:	Protection-Status
DirType$:	Typ: DIR oder FILE
DirSize&:	Programmgröße in Bytes
DirBlks&:	Programmgröße in Blocks
DirComm$:	Kommentar pro Eintrag

Variablen:

x%:	Max. Einträge
max%:	SUB-Var = x%
loop%:	Schleifenvariable
counter%:	Nummer der Einträge im Directory
fl:	Error-Flag
AccessRead%:	=-2
dir$:	Name des Directories
bytes&:	Buffer-Größe
lock&:	System-Lock für Directory
opt&:	Speicher-Options
info&:	Anfangsadresse Info-Block
suc&:	DOS-Funktion Error-Flag
DirName&:	Adresse des DIR-Namens
search:	Schleifenvariable
check:	ASCII-Zwischenspeicher
check$:	Prüfzeichenkette
prot&:	Protection-Bits
prot$:	Protection-Attribute
type&:	Typ-Bits

Programmbeschreibung:

Aus dem Directory-Infoblock sollen Name, Schutzstatus, Typ, Größe und Kommentar eines jeden Eintrages gelesen werden. Hierfür werden Variablenfelder angelegt. Die maximale Anzahl hängt von x% ab. Das Hauptprogramm main: liest das Hauptdirectory des ersten Laufwerks aus (Beispiel).

```
GetDir:
```

Die Variablen counter% und fl werden dem Hauptprogramm zugänglich gemacht, denn sowohl I/O-Fehler als auch Anzahl der gefundenen Elemente muß an dieses übergeben werden. Das Directory mit dem gewünschten Namen wird durch DosLock angepeilt. Falls es existiert (fl=0), wird ein bytes&-großer (260 Bytes) Buffer im Speicher angelegt. DosExamine untersucht den ersten Directory-Eintrag. Die benötigten Informationen werden in agin: direkt aus dem Infoblock ausgelesen.

DosExNext untersucht alle weiteren Einträge. Trifft diese Funktion auf ein NO MORE ENTRIES-Error, wird die SUB-Routine nach Zurückgabe des Buffers und Entfernen des Saugfußes verlassen.

Der Infoblock in info& hat folgendes Format:

```
00 - 03     L   Disk-Schlüssel
04 - 07     L   Art des Bufferinhaltes(größer als 0: Dir)
08 - 115        Name des Eintrages
                Bis jetzt sind lediglich 30 der
                118 Bytes nutzbar.
116 - 119   L   Schutz-Bits
                Bit   Bedeutung
                0     Delete
                1     Execute
                2     Write
                3     Read
120 - 123   L   Art des Eintrages (größer als 0=Dir)
124 - 127   L   Bytes in File
128 - 131   L   Blöcke in File
132 - 135   L   Tage seit 1. Jan 1978
136 - 139   L   Minuten seit Mitternacht
140 - 143   L   Ticks seit letzter Minute
144 - 259       Kommentar, bisher nur 80 Bytes nutzbar
```

Bemerkungen:

Wollen Sie mehr als 100 Einträge pro Directory untersuchen, so brauchen Sie lediglich x% am Anfang des Programms zu verändern. Nach Aufruf der Routine GetDir werden die gefundenen Einträge in den Variablenfeldern zu finden sein (wenn fl=0). counter% enthält die Anzahl der gefundenen Einträge. Falls counter%=x%=100 sein sollte, dann liegt der Verdacht nahe, daß sich mehr als die 100 abgelegten Einträge im Directory befinden; vergrößern Sie dann das Variablenfeld - oder tun Sie es von vornherein, wenn Sie über genügend Speicherplatz verfügen (x%=300 wird wohl allen Ansprüchen gerecht werden).

fl=1: Das angegebene Directory existiert nicht (Sie müssen den Namen immer vom Hauptdirectory her angeben, z.B. Workbench:fonts/sapphire für das Directory "sapphire".

fl=2: Es konnte kein Info-Block angelegt werden, weil kein freies RAM mehr auffindbar war.

fl=3: I/O-Error

Das folgende Programm "GetTree" zeigt eine Anwendung der GetDir-Routine.

2.7.7 GetTree - den Datenbaum untersuchen

Bekanntlich können AmigaDOS-Disketten mit verschiedenen Unter-Directories versehen werden. Das Hauptdirectory "Workbench:" enthält beispielsweise das Unterdirectory "fonts", in dem die System-Fonts gespeichert sind. Dort gibt es wiederum ein Directory namens "Sapphire", welches die verschiedenen Sapphire-Fonts beherbergt. So könnte sich der "Baum" (von einem solchen spricht man nämlich; er ähnelt einer Ahnengalerie) beliebig weiter nach unten verästeln (Schleifen von einem "tieferen" zu einem "höheren" Directory wie in UNIX sind mit AmigaDOS nicht möglich).

Diese Fülle an Directories (Inhaltsverzeichnissen) erhöht die Übersichtlichkeit eines Disketten-Inhaltsverzeichnisses ganz enorm. Aber haben Sie sich schon einmal ein solches Inhaltsverzeichnis (z.B. durch FILES) angeschaut? Die Files des aktuellen Directories werden auf den Bildschirm ausgegeben, Unterdirectories werden jedoch nur genannt, nicht ausgegeben:

```
(...)

Freddy
Freddy.info
AmigaBASIC
AmigaBASIC.info
(Unterdirectory)
Hans

(...)
```

Das folgende Programm untersucht alle Files einer Diskette, ohne Rücksicht auf Directories. Die gesamte Baumstrukur wird dabei abgeklappert. Zunächst werden die Files des Hauptdirectories ausgegeben, danach sämtliche darin vorkommende Unterdirectories, danach sämtliche in diesen vorkommende Unter-Directories, und so fort.

Der Übersicht wegen wird außerdem jeweils der Name des gerade ausgegebenen Directories mitgeliefert. Außerdem wird Programmgröße in Bytes und Blocks sowie Eintragstyp (FILE oder DIRECTORY) ausgegeben. Die Einträge werden der Übersicht wegen in alphabetischer Reihenfolge geordnet.

Das folgende Programm ist für die Ausgabe der Liste auf einen Drucker geschrieben worden, denn eine Bildschirm-Ausgabe ist nur wenig übersichtlich. Durch leichte Änderungen (LPRINTs in PRINTs verwandeln) können Sie natürlich auch ohne Drucker auskommen.

Programm-Größe: 9059 Bytes
Bemerkungen: "exec.bmap" und "dos.bmap" müssen sich auf Disk befinden

```
'GetTree

'*****************************************
'*                                       *
'* (C) 1986 by DATA BECKER GmbH W.-Ger.  *
'* (P) by Tobias "Kutte" Weltner         *
'*                                       *
'* GetTree verwendet das CLI, um die Baum- *
'* struktur einer beliebigen Diskette zu *
'* untersuchen. Das Ergebnis wird wahlwei- *
'* se auf Drucker oder Bildschirm ausgege- *
'* ben.                                  *
'*                                       *
'*****************************************


DECLARE FUNCTION DosExamine& LIBRARY     'Directory finden
DECLARE FUNCTION DosExNext& LIBRARY      'Files suchen
DECLARE FUNCTION DosLock& LIBRARY        'Directory/File Zug
  riff
DECLARE FUNCTION AllocMem& LIBRARY       'Memory besorgen
DECLARE FUNCTION DosIoErr& LIBRARY       'Fehler ausgeben

LIBRARY "exec.library"                   'da kommen all dies
  e
LIBRARY "dos.library"                    'Funktionen her


arrays:                                  'Felder
x%=200                                   'max. 200 Files pro
   Directory
y%=100                                   'max. 100 Directori
  es pro Disk
DIM SHARED DirName$(x%)
DIM SHARED DirProt$(x%)
DIM SHARED DirType$(x%)
DIM SHARED DirSize&(x%)
DIM SHARED DirBlks&(x%)
```

```
DIM a$(y%)
DIM a&(y%)

var:                                         'Variablen
filler$="."                                  'Fuell-$ fuer Ausdr
  uck
count%=1
fil$="_"                                     'Fuell-$ fuer Scree
  n

main: '==================================
                                        '=
GOSUB Request                           '=
GOSUB Header                            '=
GOSUB Level                             '=
GOSUB printTree                         '=
GOSUB printEnd                          '=
                                        '=
LIBRARY CLOSE                           '=
END                                     '=
                                        '=
'========================================

Request: '------------------------------------------------

LINE INPUT "Welches Disk-Drive (0-3) ? ----> ";Dr$

Dr%=VAL(RIGHT$(Dr$,1))
Dr$=RIGHT$(STR$(Dr%),1)
a$(0)="df"+Dr$+":"

RETURN

Header: '-------------------------------------------------

LPRINT "*  B A U M S T R U K T U R   D E C O D E R  *"
LPRINT
LPRINT "(C) 1986 by DATA BECKER GmbH/AMIGA Tips&Tricks"
LPRINT
LPRINT
```

```
RETURN

Level:'----------------------------------------------------

LPRINT STRING$(70,"_")
LPRINT
LPRINT "Level : ";Level%

RETURN

printEnd:'-------------------------------------------------

LPRINT STRING$(70,"_")
LPRINT
LPRINT "by BAUMSTRUKTUR DECODER, (C) 1986 DATA BECKER GmbH"
LPRINT STRING$(70,"_")

LOCATE 1,1
PRINT "Programm beendet.";STRING$(60,fil$)
RETURN

printTree:'------------------------------------------------


FOR loop%=previous% TO count%-1

  IF a&(loop%)=Level% THEN       'Eintrag fuer diesen Level

    search$=a$(loop%)            'das gewuenschte Directory
    GetDir search$,x%            'finden und laden
    Sort

    LOCATE 1,1
    PRINT "Printing ";a$(loop%);STRING$(60,fil$)

    IF loop%=0 THEN
      a$(loop%)=DirName$(0)+":"
    END IF
```

```
    directory$=LEFT$(a$(loop%),LEN(a$(loop%))-1)

    LPRINT
    LPRINT "Directory: ---> ";directory$

    FOR show%=1 TO counter%
      info$=DirName$(show%)

      FOR fill%=LEN(DirName$(show%)) TO 32
        info$=info$+filler$
      NEXT fill%
      IF DirType$(show%)<>"DIR " THEN

        info$=info$+DirType$(show%)
        info$=info$+"."+DirProt$(show%)
        FOR fill%=LEN(DirProt$(show%)) TO 3
          info$=info$+filler$
        NEXT fill%
        LPRINT "- ";info$;
        LPRINT USING "#######";DirSize&(show%);
        LPRINT " Bytes, ";
        LPRINT USING "###";DirBlks&(show%);
        LPRINT " Blocks."
      ELSE
        fl=1
        LPRINT "- "+info$+"DIRECTORY"
        a$(count%)=a$(loop%)+DirName$(show%)+"/"  'Name des D
  irectories
        a&(count%)=Level%+1                       'sowie Leve
  l merken
        count%=count%+1
      END IF
    NEXT show%                                    'naechster
  Eintrag in
                                                  'in diesem
  Directory
    END IF
NEXT loop%                                        'naechst. D
  irectory in
                                                   'in diese
```

```
m Level
previous%=loop%                                     'Listen-Zei
  ger

IF fl=1 THEN                                        'Mind. ein
  Directory im
  fl=0                                              'naechst. L
  evel
  Level%=Level%+1                                   'also naech
  ster Level
  GOSUB Level
  GOTO printTree                                    'und wieder
   los
END IF

RETURN
'-------------------------------------------------------------
  ---------------

SUB GetDir(dir$,max%) STATIC

subGTVar:                                            'Variablen

SHARED counter%,DirError&,fil$,fl                    '# gefunde
  ne Eintraege
                                                     'fl=1: Dir
  ectory existiert nicht
                                                     'fl=2: Kei
  n freier Speicherplatz
                                                     'fl=3: I/O
  -Error
counter%=0                                           'fuer's Ha
  uptprogramm

LOCATE 1,1
PRINT "Getting Directory ";dir$;STRING$(60,fil$)

dir$=dir$+CHR$(0)
lock&=DosLock&(SADD(dir$),-2)                       'Directory
  anpeilen
```

```
                                                  '-2=SHARED_
  ACCESS
IF lock&=0 THEN                                   'ist gar ni
  cht da...!
  PRINT "ERROR: Directory existiert nicht."
  fl=1
  GOTO ende
END IF

opt&=2^1+2^16                                     'MEMF_CLEAR
  |MEMF_CHIP
info&=AllocMem&(252,opt&)                         'info-Block
   festlegen

IF info&=0 THEN                                   'Mist, Spei
  cher alle...
  PRINT "Kein Speicherplatz frei!"
  fl=2
  GOTO ende
END IF

suc&=DosExamine&(lock&,info&)                     'Dir.'s 1.
  Eintrag
IF suc&=0 THEN                                    'Probleme..
  .
  PRINT "ERROR: Directory nicht lesbar."
  fl=3
  GOTO finish
END IF

again:                                             'Auslese-S
  chleife

DirName&=info&+8                                   'Basis-Adr
  esse String

FOR search=0 TO 29                                 'Feld:Name
   aus info-
  check=PEEK(DirName&+search)                      'Block les
  en (<31)
```

```
  IF check<>0 THEN                                   'schon vor
  her fertig?
    check$=check$+CHR$(check)                        'nein, noc
  h Zeichen
  ELSE                                               'ja, Zeit
  sparen...
    search=29                                        '...Schlei
  fe beenden
  END IF
NEXT search

DirName$(counter%)=check$:check$=""                  'Lese-Var.
   loeschen
prot&=PEEKL(info&+116)                               'Protectio
  n-Bits

IF prot&<>0 THEN                                     'jawoll, s
  ind da:
  IF (prot& AND 2^3)<>0 THEN prot$=prot$+"r"          'READ-Pr
  otected
  IF (prot& AND 2^2)<>0 THEN prot$=prot$+"w"          'WRITE-P
  rotected
  IF (prot& AND 2^1)<>0 THEN prot$=prot$+"e"          'EXECUTE
  -Protected
  IF (prot& AND 2^0)<>0 THEN prot$=prot$+"d"          'DELETE-
  Protected
  DirProt$(counter%)=prot$
  prot$=""
END IF

type&=PEEKL(info&+120)                                'Eintrags
  -Typ
IF type&<0 THEN                                       'Programm
  -FILE
  DirType$(counter%)="FILE"
ELSEIF counter%=0 THEN                                'aktuelle
  s Directory
  DirType$(counter%)="CURR.DIR"
ELSE                                                  'Director
  y naechster Level
```

```
  DirType$(counter%)="DIR "
END IF

DirSize&(counter%)=PEEKL(info&+124)                     'Programm
  -Bytes
DirBlks&(counter%)=PEEKL(info&+128)                     'Programm
  -Blocks

suc&=DosExNext&(lock&,info&)                            'naechste
  r Eintrag

IF suc&=0 THEN                                          'Fehler..
  .(na?)
  e&=DosIoErr&                                          'Meldung
  auslesen
  IF e&<>232 THEN                                       'Oh Gott,
   WIRKLICH
    PRINT "Fehler im Directory. I/O-Error Nr. ";e&      'ein Fehl
  er!
    DirError&=e&
  END IF
  GOTO finish                                           'beenden
ELSE                                                    'kein Feh
  ler
  counter%=counter%+1                                   'naechste
  r Eintrag
  IF counter%<=max% THEN                                'genug Sp
  eicher?
    GOTO again                                          'ja
  END IF
END IF                                                  'nein


finish:
CALL FreeMem(info&,252)                                 'Memory f
  reigeben
CALL DosUnLock(lock&)                                   'Director
  y befreien

END SUB
```

```
'--------------------------------------------------------------
  ----

SUB Sort STATIC                                          'Director
  y sortieren

SHARED counter%,fil$

LOCATE 1,1
PRINT "Sorting ";DirName$(0);STRING$(60,fil$)

FOR sort1=1 TO counter%                                   'Bubble-So
  rt
  FOR sort2=sort1+1 TO counter%-1
    IF UCASE$(DirName$(sort1))>UCASE$(DirName$(sort2)) THEN
      SWAP DirName$(sort1), DirName$(sort2)
      SWAP DirProt$(sort1), DirProt$(sort2)
      SWAP DirType$(sort1), DirType$(sort2)
      SWAP DirSize&(sort1), DirSize&(sort2)
      SWAP DirBlks&(sort1), DirBlks&(sort2)
    END IF
  NEXT sort2
NEXT sort1

LOCATE 1,1
PRINT "READY.";STRING$(70,fil$)

END SUB
```

Variablen-Felder:

DirName$:	Name des Eintrages
DirProt$:	Schutzstatus des Eintrags
DirType$:	Typ des Eintrages

DirSiye&:	Größe in Bytes, falls Programm
DirBlks&:	s.o., jedoch in Blocks a 512 Bytes
a$:	vollständiger Directory-Name
a&:	entsprechender Directory-Level

Variablen:

x%:	Maximal x% Einträge pro Directory
y%:	Maximal y% Directories pro Disk
filler$:	Füll-Zeichen
fil$:	zweites Füllzeichen
count%:	Directory-Zähler
Dr$:	Disk Drive
Dr%:	Nummer des Disk Drives
loop%:	Schleifenvariable
previous%:	vorheriger Schleifenwert
Level%:	Directory-Level
search$:	aktuelles Directory
directory$:	Name des Directories
show%:	Anzeige-Schleifenvariable
counter%:	Anzahl Einträge pro aktuelles Directory
info$:	Informationsangabe pro Eintrag
fl:	Flag
DirError%:	I/O-Error Nummer
dir$:	Name des gesuchten Directories (SUB)
lock&:	System-Lock des Directories
opt&:	Speicher-Optionen
info&:	Adresse des Info-Blocks
suc&:	Fehlerflag der DOS-Funktion
check:	Zeichenspeicher
check$:	entsprechendes Zeichen
search:	Suchschleifen-Variable
prot&:	Protection-Bits
type&:	Typ-Bits
e&:	I/O-Error
sort1:	Sortierschleife 1
sort2:	Sortierschleife 2
max%:	SUB 1, =x%

Programmbeschreibung:

Im ersten Teil werden alle nötigen Exec- und DOS-Funktionen deklariert. Die Variablenfelder werden initialisiert. Das Hauptprogramm besteht aus fünf Einzelprogrammen:

Request:

Das gewünschte Disk Drive wird erfragt (0-3). Der aktuelle Wert wird in Dr% als Zahl abgelegt. Das erste Directory wird in a$(0) als dfx: (x=0-3) definiert.

Header:

Ein netter Briefkopf wird gedruckt

Level:

Unter eine durchgezogene Linie wird der augenblickliche Directory-Level gedruckt.

printTree:

In der Schleife loop% werden alle neuen Einträge in a$(x) untersucht. Dies ist anfangs lediglich dfx:. Die SUB-Funktion GetDir liest das entsprechende Directory ein und initialisiert die entsprechenden DirXXX()-Felder. Diese werden anschließend durch Sort in die richtige Reihenfolge gebracht. Falls loop% = 0 ist (allerexster Eintrag), dann wird der gefundene Directory-Name ausgegeben (schließlich soll der Name der Disk ausgedruckt werden, nicht dfx:). a$(loop%) enthält den Namen des aktuellen Directories, jedoch mit einem 0-Byte am Ende. Dies wird abgezwackt und der Name ausgegeben. Die show%-Schleife gibt nun alle durch GetTree erhaltenen und Sort sortierten Einträge des aktuellen Directories aus. Anschließend wird previous% auf das Ende des bisherigen Datenblocks in a$() gesetzt, damit es beim nächsten Durchlauf bereits auf "frische" Daten zeigt.

Falls fl = 1 ist, sich also im nächsten Level noch mindestens ein Directory versteckt, wird das nächste Directory ausgegeben.

printEnd:

Eine nette Abschiedsmeldung wird gedruckt.

SUB 1: GetDir

Dieses SUB verlangt den Namen des Directories sowie die maximale Anzahl der Einträge. Diese hängt lediglich von x%, also der Größe der DirXXX()-Felder ab.

SUB 2: Sort

Nach dem Bubble-Sort-Verfahren werden die Directory-Einträge in die richtige (alphabetische) Reihenfolge gebracht.

2.7.8 ReadFile - Zuverlässiger als AmigaBASIC

Haben Sie sich schon einmal ein Datenfile angeschaut? Vielleicht so:

```
OPEN "datenfile" FOR INPUT AS 1
WHILE EOF(1)=0
  INPUT#1,in$
  PRINT in$
WEND
CLOSE 1
```

Das klappt auch ganz prima. Solange jedenfalls, bis Nullen ins Spiel kommen. Besteht Ihr Datensatz nämlich beispielsweise aus den Zahlen

0,250,34,0,0,45,0,67

so würde obiger Ausleseversuch glatt fehlschlagen. Das Ergebnis wäre:

250,34,45,67

Nullen sind weg. Warum? Tja, ...

Nun könnte man über dieses Manko großzügig hinwegsehen, denn wann braucht man schon mal Nullen?! Um jedoch echte Datenfiles (kein Text) auslesen zu können, bedarf man einer zuverlässigen Ausleseroutine, die die Nullen in Frieden läßt. Sie heißt "ReadFile" und wird so aufgerufen:

```
ReadFile "filename", "wieviel", "ab wo"
```

filename:	Der Name des Datenfiles natürlich
wieviel:	Anzahl der Bytes, die Sie auslesen wollen
ab wo:	Offset vom Datenanfang

Programm-Größe: 1420 Bytes
Bemerkungen: "dos.library"

```
'ReadFile

DECLARE FUNCTION DosOpen& LIBRARY
DECLARE FUNCTION DosRead& LIBRARY
DECLARE FUNCTION DosSeek& LIBRARY

LIBRARY "dos.library"

main:

ReadFile "t.3",1200&,0&
'                      !     !
'      wieviel Zeichen     ab wo?

'eine moegliche, wenngleich sinnlose Anwendung:

lx=LEN(code$)
FOR length=1 TO lx
  cx=cx+1
    char$=MID$(code$,length,1)
    IF char$=CHR$(10) THEN cx=0
    PRINT char$;
  IF cx>60 THEN cx=0:PRINT
NEXT length

LIBRARY CLOSE
END
```

```
'----------------------------------------

SUB ReadFile(file$,bytes&,position&) STATIC

subRFVar:                     'Variablen
SHARED code$,gelesen&,fl      'code$: Inhalt des Files
                              'gelesen&: wirklich gelesene Byte
  s
                              'fl: 1=File nicht gefunden
ModeOldFile%=1005             'File existiert bereits
offsetBeginning%=-1           'Position relativ ab File-Start

code$=SPACE$(bytes&+10)       'code$ erhaelt notwenige Laenge
file$=file$+CHR$(0)           'file$ wird mit '0' abgeschlossen


OpenFile:

'Das gewuenschte File wird geoeffnet

handle&=DosOpen&(SADD(file$),ModeOldFile%)
IF handle&=0 THEN
  PRINT "File gibt es nicht"
  fl=1
  EXIT SUB
END IF
```

```
SetPosition:

oldPos&=DosSeek&(handle&,position&,offsetBeginning%)

ReadIt:

gelesen&=DosRead&(handle&,SADD(code$),bytes&)
'handle: DOS-File Handle fuer geoeffnetes File
'SADD(code$): Anfangsadresse des Buffers (code$)
'bytes&: Laenge des Buffers

CloseFile:

CALL DosClose(handle&)


END SUB
```

Variablen:

file$:	Name des Files
bytes&:	Anzahl der gewünschten Zeichen
position&:	Zeichenoffset vom Datenanfang
code$:	ausgelesener Datenstring
gelesen&:	Anzahl der tatsächlich gelesenen Daten
fl:	Fehlerflag
ModeOldFile%:	=1005 offsetBeginning: =-1
handle&:	Filehandle des Datenfiles
oldPos&:	alter Offset

Programmbeschreibung:

Diese Unterroutine benötigt gleich drei DOS-Routinen:

DosOpen öffnet das gewünschte Datenfile und entspricht dem AmigaBASIC-Befehl OPEN. Während OPEN eine einfache Bezugszahl verlangt (z.B. OPEN "test" FOR OUTPUT AS 1; Bezugszahl ist also 1), liefert DosOpen eine sogenannte File-Handle zurück:

```
handle&=DosOpen&(SADD(file$),mode%)
```

Inputs:

file$: Name des Files; mit CHR$(0) abzuschließen.
mode%: MODE OLDFILE (=1005) öffnet bestehendes File
MODE NEWFILE (=1006) kreiert neues File

Outputs:

```
handle&  : BPTR(!) zu Handle-Datastruktur
```

Während man also in AmigaBASIC das obige File durch CLOSE 1 schließt, geschieht dasselbe in AmigaDOS durch CALL DosClose(fileHandle).

DosRead liest einen beliebig langen Datensatz aus einem zuvor geöffneten File. Das Format sieht so aus:

```
gel&=DosRead&(handle&, buffer&, bytes&)
```

Inputs:

handle&: Die Filehandle des Files, von DosOpen geliefert
buffer&: Anfangsadresse eines Speichers für die Daten.
bytes&: Die Größe dieses Buffers

Outputs:

gel&: Anzahl der Bytes, die tatsächlich gelesen wurden

DosSeek verschiebt den internen "Zeiger" in einem Datenfile. Wenn Sie ein neues File kreieren, dann ist dieser Zeiger auf den Datenanfang gerichtet. Ergänzen Sie ein File, z.B. mit APPEND, dann zeigt er auf das Ende des alten Datensatzes. DosSeek kann diesen Zeiger in beliebigen Byte-Offsets im File hin- und herverschieben. Das Format des Befehls ist:

```
oldPos&=DosSeek&(handle&,pos&,mode%)
```

Inputs:

handle&: Filehandle des Files, siehe DosOpen
pos&: Der gewünschte Offset in Bytes
mode%: OFFSET BEGINNING (=-1) Offset ab Datastart
OFFSET CURRENT (=0) Offset ab aktueller Position
OFFSET END (=1) Offset rückwärts vom Datenende

Outputs:

oldPos&: Die aktuelle Position

Anmerkung: Um die aktuelle "Zeiger"-Position zu erfahren, benutzen Sie einfach Mode OFFSET CURRENT mit einem Offset von 0.

Die Variablen code$, gelesen& und fl werden mit dem Hauptprogramm geteilt, an das sie ja letztendlich überliefert werden. Die oben näher beschriebenen Konstanten werden deklariert. Als Datenbuffer fungiert code$. Dazu muß die Stringvariable aber erst mit entsprechend vielen Leerzeichen vollgepumpt werden, an deren Stelle später die Datenbytes kommen. Das File wird geöffnet. Falls es dabei Probleme gibt (handle&=0), wird das Fehlerflag fl gesetzt. Die in position& gespeicherte Position des "Datenzeigers" wird durch DosSeek an die richtige Stelle gebracht. Anschließend wird das File mittels DosRead ausgelesen, das File geschlossen und SUB verlassen.

Anmerkung: Die Daten werden von ReadFile in code$ an das Hauptprogramm übergeben. Sie sind also als ASCII-Codes abgespeichert. Mit Hilfe der Funktion ASC dürfte die Umwandlung in Daten jedoch kein Problem sein:

```
FOR loop%=1 TO LEN(code$)
  check$=MID$(code$,loop%,1)
  PRINT ASC(check$)
NEXT loop%
```

Die Variable gelesen& gibt darüber Aufschluß, wieviel Bytes tatsächlich gelesen worden sind. Versuchen Sie beispielsweise,

aus einem 50 Bytes langen Datensatz 1000 Bytes herauszulesen, dann werden Sie trotzdem nicht mehr als 50 Bytes bekommen. gelesen& enthält dann den Wert 50. Ist gelesen& also kleiner als bytes&, dann haben Sie das Ende des Datensatzes erreicht.

2.8 I/O - Kommunikation mit der Außenwelt

Ein Ziel bei der Entwicklung des Amigas war die Schaffung einer geräte-unabhängigen I/O (Eingabe/Ausgabe; Kommunikation des Amigas mit Peripheriegeräten wie Drucker und Disk Drive). Dies erscheint jedoch schon auf den ersten Blick als unmöglich, sogar für einen Amiga: Wie kann man ein Disk Drive genau wie einen Joystick behandeln? Es geht einfach nicht.

Deshalb wurde eine I/O geschaffen, die ein Spektrum verschiedenster Geräte auf möglichst ähnliche Weise versorgt. Dazu wurden Standardstrukturen geschaffen, die für alle I/O gelten, I/O-Befehle, die für die meiste I/O anwendbar sind, sowie ein Mechanismus, mit dem gerätetypische Merkmale und Informationen in einem möglichst kompatiblen Verfahren eingeschlossen werden können.

Alle Software-Module, die I/O möglich machen, werden "devices" genannt. Es gibt beispielsweise das timer-, trackdisk-, gameport-, keyboard-, console- und audio-device, die sich alle im schreibgeschützten Kickstart-Speicher befinden, sowie narrator-, serial-, parallel-, printer- und clipboard-devices, die bei Bedarf nachgeladen werden (von der Workbench Disk).

Um I/O zu betreiben, ist es nötig, einen sogenannten I/O-Request Block zu initialisieren, denn schließlich muß Amiga wissen, was Sie wollen. Dieser Block sieht folgendermaßen aus:

```
IOStdReq: (I/O Standard Request Block)
________________________________________
00 - 03  L  Node * ln Succ
04 - 07  L  Node * ln Pred
08       B  Type
```

09		B	Priorität
10 - 13		L	* ln Name
14 - 17		L	MsgPort * mn ReplyPort
18 - 19		W	mn Length
20 - 23		L	Device *io Device
24 - 27		L	Unit *io Unit
28 - 29		W	io Command
30		B	Flags
31		B	Error
32 - 35		L	Actual
36 - 39		L	Length
40 - 43		L	Data
44 - 47		L	Offset

Der erste Teil besteht aus einer einfachen Node. Was das ist, erklären wir gleich. Zusammen mit dem zweiten Teil bildet sie die C-Struktur io Message. Neben dieser Struktur muß ein sogenannter Message Port eingerichtet werden (seine Anfangsadresse gehört ins Feld "MsgPort * mn ReplyPort"). Der Message Port baut sich so auf:

<u>MsgPort:</u>

00 - 03	L	Node * ln Succ
04 - 07	L	Node * ln Pred
08	B	Type
09	B	Priorität
10 - 13	L	* ln Name
14	B	mp Flags
15	B	mp SigBit
16 - 19	L	Task * mp SigTask
20 - 23	L	Node * lh Head
24 - 27	L	Node * lh Tail

```
28 - 31  L  Node * lh TailPred
32       B  lh Type
33       B  lh Pad (nutzlos)
```

Der erste Teil ist wieder eine Node-Struktur. Der mittlere Teil bildet das Herz des Message Ports, und der letzte Teil ist eine List-Struktur.

Der Sinn und Zweck einer I/O-Struktur ist ja noch einzusehen, denn dort kann man angeben, was wie gemacht werden soll. Aber wofür braucht man einen Message Port, ganz zu schweigen von den blöden Nodes?

Der Amiga ist ein Multi-Tasking Computer. Es können also scheinbar mehrere Programme gleichzeitig und unabhängig voneinander laufen. Der Prozessor sorgt bei diesem Verfahren dafür, daß alle Programme nacheinander Rechenzeit zugewiesen bekommen. Zunächst läuft das Programm mit höchster Priorität. Muß dieses Programm auf etwas warten (einen Tastendruck des Benutzers, beispielsweise), oder ist das Zeitlimit (ein paar Mikrosekunden) abgelaufen, dann kommt das Programm mit der nächstniedrigeren Priorität zum Zuge, während das erste Programm auf Eis gelegt wurde.

Aus der Sicht des einzelnen Programmes wird diese Zeitzuweisung gar nicht bemerkt (ein Programm merkt ja bekanntlich nicht, daß es abgeschaltet worden ist), und läuft also scheinbar gleichzeitig mit vielen anderen Programmen.

Möchte ein Programm jedoch mit einem zweiten, "gleichzeitig" laufenden, kommunizieren, dann wird es schon schwieriger. Da in Wirklichkeit ja immer nur ein Programm läuft, ist es schwierig, an ein anderes, auf Eis gelegtes Programm, Daten zu senden. Daher bedient man sich des Message-Ports, mit denen einzelne Tasks (Programme) versehen werden können. Ein Message-Port ist eine Art Briefkasten, in den andere Programme Daten legen können, die dann vom Zielprogramm gelesen werden, sobald es wieder einmal Rechenzeit bekommt.

Die Node-Strukturen können Sie als Benutzer ganz kalt lassen, denn Sie brauchen sich nicht um sie zu kümmern. Das System benutzt sie, um Listen zu bilden. Dabei ist die Node-Struktur sowas wie ein Mitgliedskopf, der Informationen über das vorangegangene und noch folgende Mitglied enthalten. Außerdem kann jede Node mit einem Namen und einer Priorität versehen werden. Oftmals werden dann Listen mit nach Priorität geordneten Nodes geschaffen.

2.8.1 AmigaBASIC und I/O

Nach soviel grauer Theorie endlich wieder etwas praktisches! Vielleicht haben die vielen Strukturen Sie etwas abgeschreckt. Alles ist halb so kompliziert. Schauen Sie sich einmal die folgenden SUB-Routinen an. Sie erledigen die gröbsten Initialisierungsaufgaben für Sie:

Programm-Größe: 2146 Bytes
Bemerkungen: "exec.bmap" muß sich auf Disk befinden

```
'I/O-Packet

DECLARE FUNCTION OpenDevice& LIBRARY
DECLARE FUNCTION CloseDevice& LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION AllocSignal& LIBRARY
DECLARE FUNCTION FindTask& LIBRARY
DECLARE FUNCTION DoIO& LIBRARY


LIBRARY "exec.library"  'exec.library eroeffnen

CreatePort
CreateIO port&
OpenDevice "trackdisk.device",io&,0&

PRINT "Disk 0: Motor Test - Status: xxx"
LOCATE 1,1

FOR t=1 TO 10000:NEXT t   'motor auslaufen
```

```
PRINT TAB(30);"ein"
POKEW io&+28,9
POKEL io&+36,1
e&=DoIO&(io&)

PRINT "Maus-Taste betaetigen!"

SLEEP
SLEEP

LOCATE 1,1
PRINT TAB(30);"aus"
PRINT "Test beendet."+SPACE$(30)

POKEL io&+36,0
e&=DoIO&(io&)

ShutDevice io&
RemovePort port&,sigBit%
DeleteIO io&

LIBRARY CLOSE
END


LIBRARY CLOSE  'exec.library wieder schliessen
END

'-----------------------------------------------------

SUB OpenDevice (dev$,io&,unit&) STATIC

dev$=dev$+CHR$(0)
succ&=OpenDevice&(SADD(dev$),unit&,io&,0)
IF succ&<>0 THEN
  PRINT "Device laesst sich nicht oeffnen."
  EXIT SUB
END IF

END SUB

'-----------------------------------------------------

SUB ShutDevice(io&) STATIC

CALL CloseDevice(io&)

END SUB
```

```
-----------------------------------------------------

SUB CreatePort STATIC

SHARED port&,sigBit%
opt&=2^0+2^16
port&=AllocMem&(38,opt&)
IF port&=0 THEN PRINT "Kein Speicherplatz!":EXIT SUB
sigBit%=AllocSignal&(-1):IF sigBit%=-1 THEN ERROR 255
sigTask&=FindTask&(0)

POKE  port&+8,4
POKEL port&+10,port&+34
POKE  port&+15,sigBit%
POKEL port&+16,sigTask&
POKEL port&+20,port&+24
POKEL port&+28,port&+20
POKE  port&+34,ASC("M")
POKE  port&+35,ASC("S")
POKE  port&+36,ASC("G")
CALL AddPort(port&)

END SUB

-----------------------------------------------------

SUB RemovePort(port&,sigBit%) STATIC

CALL RemPort(port&)
CALL FreeSignal(sigBit%)
CALL FreeMem&(port&,38)

END SUB
```

```
'-----------------------------------------------------

SUB CreateIO(port&) STATIC

SHARED io&
opt&=2^0+2^16
io&=AllocMem&(62,opt&)
IF io&=0 THEN PRINT "Kein Speicherplatz!":EXIT SUB
POKE  io&+8,5
POKEL io&+14,port&
POKEW io&+18,12

END SUB

'-----------------------------------------------------

SUB DeleteIO(io&) STATIC

CALL FreeMem(io&,62)

END SUB
```

Variablen:

e&:	I/O-Error Flag
dev$:	Name des Devices
io&:	Adresse des I/O-Request Blocks
unit&:	Nummer der Einheit
succ&:	Fehlerflag der exec-Funktion
port&:	Adresse des Message Ports
opt&:	Memory-Optionen
sigBit%:	Signalbit
sigTask&:	Adresse des Taskhandlers

Programmbeschreibung:

Diese Routinensammlung enthält alle Funktionen, die für die Nutzung der I/O notwendig sind:

1. CreatePort/RemovePort

CreatePort initialisiert eine Message Port Struktur, bindet den neuen Port in die Systemliste ein und liefert die Anfangsadresse des neuen Ports an das Hauptprogramm zurück.

Die beiden Variablen port& und sigBit% werden als Öffentlich deklariert; port& beinhaltet die Anfangsadresse des neuen Ports und soll später im Hauptprogramm weiterverwendet werden können, während sigBit% für den Aufruf RemovePort zur Verfügung stehen muß.

Ein 38 Bytes großer Speicherbereich wird reserviert. Außerdem wird ein Signalbit gepachtet und der Taskhandler gesucht (und gefunden). Jetzt kann die Message-Port-Struktur initialisiert werden (siehe Anfang dieses Kapitels): Node-Typ ist NT MESSAGE PORT (=4), der Name der Node "MSG". SigBit und Task werden eingesetzt, die List-Struktur initialisiert (lh Head zu lh Tail; lh TailPred zu lh Head). Anschließend wird der neue Port via exec-Befehl AddPort in die Systemliste eingebunden. Sobald AddPort aufgerufen wurde, ist der neue Port "MSG" Teil der System-Liste. Die vormals mit 0 initialisierten Node-Felder ln Succ sowie ln Pred zeigen nun auf den vorangegangenen, bzw. folgenden Port im System.

RemovePort verlangt die Adresse des betreffenden Ports und das Signalbit, und sorgt dafür, daß dem Amiga zurückgegeben wird, was während der Schaffung des Ports genommen worden war: Der Port wird zunächst aus der Systemliste gekoppelt (RemPort), das Signalbit wird wieder freigegeben (FreeSignal), und schließlich werden die durch den Port belegten 38 Bytes zurückgegeben (Kleinvieh macht auch Mist).

2. CreateIO/DeleteIO

CreateIO verlangt die Adresse eines bereits durch CreatePort initialisierten Message Ports. Es initialisiert den I/O Standard Request Block und liefert seine Anfangsadresse in io& ans Hauptprogramm zurück.

Zunächst wird io& veröffentlicht. Ein 62 Bytes großer Speicherbereich wird beschlagnahmt, in den anschließend der Request Block gelegt wird. Der eigentliche Standard Request Block benötigt übrigens nur 48 Bytes, aber es gibt einige Spezialfälle (beispielsweise für Extended Disk I/O und spezielle Druckerfunktionen; mehr darüber später), für die bis zu 62 Bytes zur Verfügung stehen müssen.

Der Typ ist NT MESSAGE (=5), Message Port ist port&, die Länge der Message ist 44.

Hier wird wieder das Prinzip des Message Ports klar: Der gesamte I/O Standard Request Block wird durch den Message-Teil zu einer Mitteilung, die an einen Message Port geschickt werden kann (den Message Port des Handlers). Gleichzeitig muß aber auch der request Block einen Message Port besitzen, an welchen der Handler eine Botschaft zurücksenden kann. Man nennt den Message Port des Blocks deshalb oft auch Reply Port (Antwort-Port). DeleteIO gibt die 62 I/O-Block Bytes wieder zurück.

3. OpenDevice/ShutDevice

OpenDevice aktiviert den Device Handler. Bereits vorher wurde angesprochen, was ein "Device" ist. Es ähnelt nicht nur inhaltlich einer Library, es muß auch wie eine solche geöffnet werden. OpenDevice fordert den Devicenamen, die Anfangsadresse des I/O Standard Request Blocks sowie eine Einheit-Nummer (normalerweise =0). Das Device wird eröffnet; es wird kein Wert ans Hauptprogramm zurückgeliefert. Der Name des Devices in dev$ wird mit einem Null-Byte abgeschlossen. Ein Ruf zur exec-Funktion OpenDevice öffnet den Handler. Dieser Handler füllt

die Felder "Device" und "Unit" im I/O-Block mit den korrekten Werten.

ShutDevice benötigt die Andresse des entsprechenden I/O-Blocks und besteht lediglich aus dem Aufruf der exec-Funktion "CloseDevice".

2.8.2 Drucker-I/O

Die einfachste Möglichkeit, mit dem Printer in Kontakt zu treten, sind die Befehle LPRINT und LLIST. Nicht viel schwieriger ist es jedoch, das DOS-Printer-Device zu eröffnen. Es genügen die folgenden Zeilen:

```
OPEN "prt:" FOR OUTPUT AS 1
PRINT#1,"Hello Sebastian!"
CLOSE 1
```

Die Amiga-Leute mußten sich ja bereits bei der Entwicklung mit dem Problem auseinandersetzen, verschiedenste Druckertypen möglichst unkompliziert ansprechen zu können. Dazu wurden die "Preferences" geschaffen, in denen man sich seinen Lieblingsdrucker einfach aussucht und der Amiga daraufhin mit diesem umzugehen weiß. Oftmals möchte man aber auch spezielle Druckereigenschaften wie Reversdruck, NLQ oder Druckmodi ändern. Damit auch dies so universell wie möglich geschehen kann, gibt es eine Liste festgesetzter Zeichenkombinationen, die für jeden Drucker gleich sind. Erst der Druckertreiber, der die genaueren Druckerparameter kennt, wandelt die Codes in die druckerspezifischen Werte um. Hier die Liste:

Reset	ESC+c
Initialisieren	ESC+#1
Line Feed	ESC+D
Return, Line Feed	ESC+E
Reverse Line Feed	ESC+M
Norm. Zeichensatz	ESC+[0m
Schrägschrift ein	ESC+[3m
Schrägschrift aus	ESC+[23m

Unterstreichen ein	ESC+[4m
Unterstreichen aus	ESC+[24m
Fettdruck ein	ESC+[1m
Fettdruck aus	ESC+[22m
Vordergrundfarbe	ESC+[nm
(n = zwei ASCII-Werte; 3 gefolgt von 0 - 9)	
Hintergrundfarbe	ESC+[nm
(n = zwei ASCII-Werte; 4 gefolgt von 0 - 9)	
Norm. Schrift	ESC+[0w
Elite ein	ESC+[2w
Elite aus	ESC+[1w
Feinschrift ein	ESC+[4w
Feinschrift aus	ESC+[3w
Breitschrift ein	ESC+[6w
Breitschrift aus	ESC+[5w
Schattenschrift ein	ESC+[6"z
Schattenschrift aus	ESC+[5"z
Doublestrike ein	ESC+[4"z
Doublestrike aus	ESC+[3"z
NLQ ein	ESC+[2"z
NLQ aus	ESC+[1"z
Hochschrift ein	ESC+[2v
Hochschrift aus	ESC+[1v
Tiefschrift ein	ESC+[4v
Tiefschrift aus	ESC+[3v
Norm.Linie	ESC+[0v
Halbschritt hoch	ESC+L
Halbschritt tief	ESC+K
US-Zeichensatz	ESC+(B
Franz. Zeichensatz	ESC+(R
Deutsch. Zeichensatz	ESC+(K
Engl. Zeichensatz	ESC+(A
Dänisch. Zeichensatz	ESC+(E
Schwed. Zeichensatz	ESC+(H
Ital. Zeichensatz	ESC+(Y
Span. Zeichensatz	ESC+(Z
Japan. Zeichensatz	ESC+(J
Norweg. Zeichensatz	ESC+(6

```
Dän. II Zeichensatz    ESC+(C
Proportional ein       ESC+[2p
Proportional aus       ESC+[1p
Proportional clear     ESC+[0p
Proport. Offset setz.  ESC+[n E
autom. Linksrand       ESC+[5 F
autom. Rechtsrand      ESC+[7 F
Blocksatz ein          ESC+[6 F
Blocksatz aus          ESC+[0 F
Justify                ESC+[3 F
Auto Center            ESC+[1 F
1/8" pro Linie         ESC+[0z
1/6" pro Linie         ESC+[1Z
```

Natürlich funktionieren diese Kommandos nur bei Drucker, die auch in der Lage sind, sie auszuführen. Die Zeilen

```
OPEN "prt:" FOR OUTPUT AS 1
PRINT#1,CHR$(27)+"[3m"
PRINT "HELLO ERNESTINUM!"
CLOSE 1
```

würde beispielsweise den Text nur dann kursiv drucken, wenn der Drucker dafür auch vorgesehen ist.

2.8.2.1 Dump - Grafik-Hardcopies

Natürlich kann man den Drucker auch via "printer.device" als I/O-Gerät durch exec ansprechen. Der Aufwand, der dabei entsteht, lohnt sich jedoch für die meisten Anwendungen nicht. "printer.device" hat jedoch einen Befehl zu bieten, der die Schwierigkeiten vergessen läßt: Ein beliebiger Rastport (Window- oder Screen-Struktur) läßt sich auf einen (grafikfähigen) Drucker ausgeben. Dabei werden die Farben in verschiedene Muster verwandelt, falls es sich nicht um einen Farbdrucker handelt.

Das folgende Programm bietet Ihnen den dump-befehl, der folgendermaßen aufgerufen wird:

```
dump "name",mode%
```

Der Name entspricht dem Namen des Windows, das Sie gerne ausgedruckt hätten. Sie können aber auch SCREENx angeben, wobei x eine Zahl zwischen 0 und ... ist und die Nummer des Screens darstellt; SCREEN0 steht z.B. für den Workbench-Screen.

Mode%=0: Die Bildschirmausdehnungen werden 1:1 in Druckerpixel umgewandelt
Mode%=1: Es wird die maximale Druckerkapazität verwendet.

Anmerkung: Es dauert seine Zeit, Grafik auszudrucken. Unidirektionaler Druck ist schneller als bidirektionaler. Im Normalfall ist die Hintergrundfarbe beim Amiga ein dunkles Blau. Diese Farbe wird bei einem Matrixdrucker natürlich als dunkles Raster mitgedruckt. Wollen Sie einen weißen Hintergrund, dann müssen Sie die Bildschirmfarben entsprechend vor dem Druck ändern.

Während des Druckens kann das programm nicht unterbrochen werden.

```
'*****************************************************
'*                                                   *
'*  (C) 1986 by DATA BECKER GmbH Duesseldorf/FRG     *
'*  (P) 15. August 1986 by Tobias "Kutte" Weltner    *
'*                                                   *
'*  Dieses Programm verwendet das "printer.device"   *
'*  des AMIGAs, um beliebige WINDOW- oder SCREEN-    *
'*  inhalte auf einem Drucker auszugeben.            *
'*                                                   *
'*  Eine detaillierte Programm-Beschreibung sowie    *
'*  die technischen Hintergruende und Anregungen     *
'*  finden Sie in DATA BECKER's "AMIGA Tips &        *
'*  Tricks".                                         *
'*                                                   *
'*****************************************************
```

```
'Definition der verwendeten Library-Funktionen

DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION OpenDevice& LIBRARY
DECLARE FUNCTION AllocSignal& LIBRARY
DECLARE FUNCTION FindTask& LIBRARY
DECLARE FUNCTION DoIO& LIBRARY

LIBRARY "exec.library"  'exec.library eroeffnen

main:'=============================================
                                                 '=
    GOSUB ChangeColors                           '=
    GOSUB DrawStuff                              '=
    GOSUB Request                                '=
                                                 '=
  LIBRARY CLOSE  'exec.library wieder schliessen '=
                                                 '=
END                                              '=
                                                 '=
'==================================================

ChangeColors:'-------------------------------------
```

```
PALETTE 0,1,1,1
PALETTE 1,0,0,0
PALETTE 2,.3,.3,.3
PALETTE 3,.7,.7,.7

RETURN

DrawStuff:'-----------------------------------------

FOR x=1 TO 10
CIRCLE (100,100),50+(2*x),1
NEXT x

RETURN

Request:'-------------------------------------------

LOCATE 1,1
PRINT "        *** DUMPER ***"
PRINT "Eine Grafik-Hardcopy-Routine"
LOCATE 5,1
PRINT "Nennen Sie den Namen des Windows..."
PRINT "z.B. 'LIST' "
PRINT "oder verwenden Sie SCREEN fuer die Aus-"
PRINT "gabe eines ganzen Bildschirms..."
PRINT "z.B. 'SCREEN0', 'SCREEN1', ..."
PRINT

LINE INPUT "Ihr Wunsch---> ";wd$

Dump wd$,1  'selbst-implementierte Dump-Funktion

RETURN

'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
SUB Dump (wind$,mode%) STATIC                          '!
                                                       '!
SHARED m$,add&,pio&,port&,sigBit%,spec%,s%

IF UCASE$(LEFT$(wind$,6))=UCASE$("screen") THEN
```

```
  s%=VAL(RIGHT$(wind$,1))
  m$="screen"
ELSE
  m$=wind$
END IF

IF mode%=0 THEN
  spec%=0        '1:1 Print
ELSE
  spec%=255      '1:MAX Print
END IF

CreatePort      'Message-Port und I/O-Struktur einrichten
SetParameters   'Parameter in I/O-Struktur einfuellen

dev$="printer.device"+CHR$(0)
suc&=OpenDevice&(SADD(dev$),0,pio&,0)    'Drucker oeffnen
IF suc&<>0 THEN                          'Drucker will nicht
  PRINT "Printer Device will nicht"      'z.B. immernoch offe
  n
  END                                    'oder gar nicht da (
  ??!)
END IF

'====================  Aktivieren der I/O
                   '=  Nur wenn alle Felder korrekt initialis
  iert sind,
e&=DoIO&(pio&)     '=  darf die I/O aktiviert werden.
                   '=  Ansonsten meldet sich der boese GURU
'====================

PRINT "Error = ";PEEK(pio&+31) 'Error Report

'---------------------------------------------------
ender:

CALL CloseDevice(pio&)        'Printer schliessen
CALL RemPort(port&)           'Message-Port entfernen
CALL FreeSignal(sigBit%)      'Signal-Bit freigeben
CALL FreeMem(pio&,62)         'I/O-Speicher und
```

```
CALL FreeMem(port&,38)          'Message-Port-Speicher
                                'zurueckgeben
END SUB

'--------------------------------------------------------
SUB CreatePort STATIC

SHARED pio&,port&,sigBit%

'printer IO einrichten

opt&=2^0+2^16    'MEMF_CLEAR|MEMF_CHIP
pio&=AllocMem&(62,opt&)  'I/O-memory-block
port&=AllocMem&(38,opt&) 'Message-Port memory-block
IF pio&=0 OR port&=0 THEN       'ERROR
  PRINT "Ueber den Nutzen einer Speichererweiterung laesst s
  ich bekanntlich"
  PRINT "streiten. An dieser Stelle waere jedoch eine solche
   Anschaffung"
  PRINT "aeusserst interessant, da Ihr AMIGA im Begriff ist,
   die letzten"
  PRINT "Speicherplatz-Reserven anzubrechen. Leider nicht ge
  nug fuer dieses"
  PRINT "Programm. (Schliessen Sie unnoetige Fenster und unn
  uetz herum-"
  PRINT "schwirrende Tasks, oder kaufen Sie sich eine Erweit
  erung. Dann"
  PRINT "koennen Sie nochmal vorbeischauen!)"
  ERROR 7
END IF

sigBit%=AllocSignal&(-1)        'Signal-Bit einsacken
IF sigBit%=-1 THEN
  forever:
  PRINT "Alle Signal-Bits sind besetzt. Bitte legen Sie nich
  t auf! Sie"
  PRINT "werden bedient, sobald das naechste Signal-Bit frei
   ist... "
  GOTO forever
END IF
```

```
sigTask&=FindTask&(0)

'---Message-Port Initialisieren------------------------------
  ----------

'Notfalls koennen alle Zeilen, die mit einem Stern (*) geken
  nzeichnet
'sind, weggelassen werden. Es handelt sich bei ihnen naemlic
  h um Fel-
'der, die entweder ohnehin re-initialisiert oder automatisch
   gefuellt
'werden.
'
'Falls Sie jedoch Zeit und Lust haben, sollten Sie sich die
  Muehe
'machen, sie mit zu uebernehmen, da es (1.) um's Prinzip geh
  t und (2.)
'moegliche spaetere Erweiterungen und Veraenderungen erhebli
  ch verein-
'facht.

POKEL port&,0              'ln_Succ (*)
POKEL port&+4,0            'ln_Pred (*)
POKE  port&+8,4            'Type=NT_MSGPORT
POKE  port&+9,0            'Priority(*)
POKEL port&+10,port&+34    'Name
POKE  port&+14,0           'Flags=PA_SIGNAL (*)
POKE  port&+15,sigBit%
POKEL port&+16,sigTask&
POKEL port&+20,port&+24    'lh_Head
POKEL port&+24,0           'lh_Tail (*)
POKEL port&+28,port&+20    'lh_TailPred
POKE  port&+32,0           'lh_Type (*)
POKE  port&+33,0           'lh_Alignment (*)
POKE  port&+34,ASC("P")    'Name
POKE  port&+35,ASC("R")    'der
POKE  port&+36,ASC("T")    'Struktur
POKE  port&+37,0           '0=ende  (*)
```

```
'--------------------------------------------

CALL AddPort(port&)        'neuen Port in System einbinden

'--------------------------------------------

POKEL pio&,0               'ln_Succ (*)
POKEL pio&+4,0             'ln_Pred (*)
POKE  pio&+8,5             'NT_MESSAGE
POKE  pio&+9,0             'Priority(*)
POKEL pio&+10,0            'kein Name (*)

POKEL pio&+14,port&
POKEW pio&+18,12           'Length of Message
POKEL pio&+20,0            'io_Device (*)
POKEL pio&+24,0            'io_Unit   (*)
POKEW pio&+28,0            'Command   (*)
POKE  pio&+30,0            'Flags     (*)
POKE  pio&+31,0            'io_Error  (*)

POKEL pio&+32,0            'Actual    (*)
POKEL pio&+36,0            'Length    (*)
POKEL pio&+40,0            'DATA      (*)
POKEL pio&+44,0            'Offset    (*)

END SUB

 --------------------------------------------

SUB SetParameters STATIC

SHARED add&,m$,pio&,spec%,s%

IF m$="screen" THEN        'Ein SCREEN soll's sein
  fl=2                     'fl=2:SCREEN, fl=0:WINDOW, fl=1:HIR
  NGESPINST
  dwindow&=WINDOW(7)
ELSEIF m$<>"" THEN                      'Sonderwuensche
  findwindow m$
  IF fl=1 THEN                          'gibt's nicht
```

```
    fl=0
    PRINT "Window Not Found!"
    GOTO ender
  END IF
  dwindow&=add&
ELSE                                    'Ein BASIC-Window, bit
  te!
  dwindow&=WINDOW(7)
END IF

  dRastPort&=PEEKL(dwindow&+50)         'RastPort fuer WINDOWs

  dScreen&=dwindow&+46                  'SCREEN Einsprungadres
  se
  FOR loop=0 TO s%                      'Adresse der gewuensch
  ten SCREEN-
    found&=PEEKL(dScreen&)              'Struktur suchen
    IF found&<>0 THEN                   '(normales Window: s%=
  0)
      dScreen&=found&                   '(normaler Screen: s%=
  0)
    ELSE                                'max. 10 Screens (9 ad
  d.)
      PRINT "SCREEN ";loop-1
      PRINT "ERROR - gewuenschter Screen nicht vorhanden!"
      loop=s%
    END IF
  NEXT loop
  dViewPort&=dScreen&+44                'ViewPort des SCREENs
  dColorMap&=PEEKL(dViewPort&+4)        'Farb-Tabelle des SCRE
  ENs
  dMode=PEEKW(dViewPort&+32)            'Darstellungsmode des
  SCREENs
  flag=PEEK(dwindow&+26)                'WINDOW-Flag;
                                        'Bit 4 = 0: normal
                                        'Bit 4 = 1: gimmezeroz
  ero
  IF (flag AND 4)<>0 THEN               'Bildschirm-Informatio
  nen fuer
    dwidth=PEEKW(dwindow&+112)          'Gimmezerozero-Window
```

```
Koordinaten
    dHeight=PEEKW(dwindow&+114)          ´holen
  ELSE                                   ´Bildschirm-Informatio
  nen fuer
    dwidth=PEEKW(dwindow&+8)             ´anderes Window holen
  (ebenfalls
    dHeight=PEEKW(dwindow&+10)           ´Koordinaten)
  END IF

IF fl=2 THEN                             ´Ein Screen (fl=2):
  dRastPort&=dScreen&+84                 ´Spezieller RastPort
  dwidth=PEEKW(dScreen&+12)              ´Spezielle Koordinaten
   (norm.
  dHeight=PEEKW(dScreen&+14)             ´x=640, y=200)
END IF

´-------------------------------------------------------------
  --------------

POKEW pio&+28,11                         ´DumpRastPort
POKE  pio&+30,1                          ´QuickIO  (*)
POKEL pio&+32,dRastPort&
POKEL pio&+36,dColorMap&
POKEL pio&+40,dMode
POKEW pio&+44,0                          ´OffsetX  (*)
POKEW pio&+46,0                          ´OffsetY  (*)
POKEW pio&+48,dwidth                     ´Quelle x
POKEW pio&+50,dHeight                    ´Quelle y
POKEL pio&+52,dwidth                     ´dest x
POKEL pio&+56,dHeight                    ´dest y
POKEW pio&+60,spec%                      ´special effects

END SUB

´-------------------------------------------------------------
  -------------

SUB findwindow(nm$) STATIC               ´geh such Int.´s WINDO
  W-Struktur
wind$=nm$
```

```
SHARED add&                                  'add& geht zurueck an
  main:

add&=WINDOW(7)                               'Einstiegspunkt

SetToStart:
  ch&=PEEKL(add&+66)                         'Zunaechst die Kette b
  is nach oben
  IF ch&<>0 THEN                             'durcharbeiten
    add&=ch&
    GOTO SetToStart
  END IF

FindTitle:                                   'Name des gefundenen W
  indows
title&=PEEKL(add&+32)                        'finden

FOR counter=0 TO 100
  check=PEEK(title&+counter)
  IF check<>0 THEN
    check$=check$+CHR$(check)
  ELSE
    counter=100
  END IF
NEXT counter

comp$=check$:check$=""

IF UCASE$(wind$)<>UCASE$(comp$) THEN         'na, das Gesuchte gefu
  nden?
  add&=PEEKL(add&+70)                        'nein
  IF add&<>0 THEN                            'aber da ist ja noch e
  ins...
    GOTO FindTitle
  ELSE                                       'alle Windows durch
    PRINT "Window gibt es nicht."
    fl=1                                     'fl=1:HIRNGESPINST
  END IF
END IF

END SUB
```

2.8.3 Trackdisk-Device

Als ein kleines Beispiel dafür, was man mit dem I/O-Packet in Beziehung Disk-Drive machen kann, versteht sich das folgende Programm. Es liest beliebige Sektoren einer Diskette und stellt den Inhalt als hexadezimale Zahlen sowie ASCII-Zeichen dar.

```
'diskEd

DECLARE FUNCTION OpenDevice& LIBRARY
DECLARE FUNCTION CloseDevice& LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION AllocSignal& LIBRARY
DECLARE FUNCTION FindTask& LIBRARY
DECLARE FUNCTION DoIO& LIBRARY
DECLARE FUNCTION OpenFont& LIBRARY

LIBRARY "exec.library"     'exec.library eroeffnen
LIBRARY "graphics.library" 'graphics.lib eroeffnen

charset:    '(80 Zeichen pro Zeile)

font0$="topaz.font"+CHR$(0)
textAttr&(0)=SADD(font0$)
textAttr&(1)=8*2^16
strFont&=OpenFont&(VARPTR(textAttr&(0)))
IF strFont&<>0 THEN CALL SetFont(WINDOW(8),strFont&)
PRINT "[R]EAD   E[X]IT"
var:

cmdRead=2
tdsector=512

main:

CreatePort
CreateIO port&
io&(0)=io&
CreateIO port&
io&(1)=io&
OpenDevice "trackdisk.device",io&(0),0&
OpenDevice "trackdisk.device",io&(1),1&

CreateBuffer 512&
diskBuffer&(0)=buffer&
CreateBuffer 512&
diskBuffer&(1)=buffer&
```

```
GOSUB editor

ClearBuffer diskBuffer&(0),512&
ClearBuffer diskBuffer&(1),512&
ShutDevice io&(0)
ShutDevice io&(1)
RemovePort port&,sigBit%
DeleteIO io&(0)
DeleteIO io&(1)

LIBRARY CLOSE
END

editor:

GOSUB Request
marker=1
IF in$="R" THEN
  LINE INPUT "Drive  > ";tin$
  drive%=VAL(tin$)
  LINE INPUT "Sector > ";tin$
  sector&=VAL(tin$)
  IF sector&>1760 THEN
    PRINT "INPUT ERROR: SECTOR 0 - 1759"
  END IF
  sector&=sector&
  command1&=2
  ChangeSector command1&,io&(drive%),sector&,diskBuffer&(dri
  ve%)
  MotorOff io&(drive%)
  FOR loop=0 TO tdsector-1 STEP 25
    FOR loop2=0 TO 24
      check=PEEK(diskBuffer&(drive%)+loop2+loop)
      a$=HEX$(check)
      IF LEN(a$)=1 THEN
        a$="0"+a$
      END IF
      he$=he$+a$
      IF check<31 THEN
```

```
        de$=de$+"?"
      ELSE
        de$=de$+CHR$(check)
      END IF
      IF loop2+loop=tdsector-1 THEN
        loop2=24
      END IF
    NEXT loop2
    PRINT he$;"   ";de$
    he$="":de$=""
    counter=counter+1
  NEXT loop
  counter=0
ELSEIF in$="X" THEN
  RETURN
END IF

GOTO editor

Request:

current=0
FOR renew=0 TO 10
  in$(renew)=""
NEXT renew

LINE INPUT "> ";in$
FOR loop=1 TO LEN(in$)
  check$=MID$(in$,loop,1)
  IF check$<>" " AND check$<>"," THEN
    fl=0
    in$(current)=in$(current)+UCASE$(check$)
  ELSE
    IF fl=0 THEN
      fl=1
      current=current+1
    END IF
  END IF
NEXT loop
in$=UCASE$(LEFT$(in$(0),1))
```

```
RETURN

'-----------------------------------------------------------

SUB ChangeSector (command&,io&,sector&,buffer&) STATIC

tdsector=512
POKEW io&+28,command&
POKEL io&+36,tdsector
POKEL io&+40,buffer&
tdoffset=sector&*tdsector
POKEL io&+44,tdoffset
e&=DoIO&(io&)

END SUB

SUB Update(io&) STATIC

POKEW io&+28,4
e&=DoIO&(io&)

END SUB

SUB MotorOff (io&) STATIC

POKEW io&+28,9
POKEL io&+36,0
e&=DoIO&(io&)

END SUB

SUB CreateBuffer (size&) STATIC

SHARED buffer&,fl

opt&=2^0+2^16
buffer&=AllocMem&(size&,opt&)
IF buffer&=0 THEN
```

```
  PRINT "Kein Speicherplatz!"
  fl=1
END IF

END SUB

SUB ClearBuffer (address&,size&) STATIC

CALL FreeMem(address&,size&)

END SUB

SUB OpenDevice (dev$,io&,unit&) STATIC

dev$=dev$+CHR$(0)
succ&=OpenDevice&(SADD(dev$),unit&,io&,0)
IF succ&<>0 THEN
  PRINT "Device laesst sich nicht oeffnen."
  EXIT SUB
END IF

END SUB

'----------------------------------------------------

SUB ShutDevice(io&) STATIC

CALL CloseDevice(io&)

END SUB

'----------------------------------------------------

SUB CreatePort STATIC

SHARED port&,sigBit%
opt&=2^0+2^16
port&=AllocMem&(38,opt&)
IF port&=0 THEN PRINT "Kein Speicherplatz!":EXIT SUB
sigBit%=AllocSignal&(-1):IF sigBit%=-1 THEN ERROR 255
```

```
sigTask&=FindTask&(0)
POKE  port&+8,4
POKEL port&+10,port&+34
POKE  port&+15,sigBit%
POKEL port&+16,sigTask&
POKEL port&+20,port&+24
POKEL port&+28,port&+20
POKE  port&+34,ASC("M")
POKE  port&+35,ASC("S")
POKE  port&+36,ASC("G")
CALL AddPort(port&)

END SUB

'-----------------------------------------------------

SUB RemovePort(port&,sigBit%) STATIC

CALL RemPort(port&)
CALL FreeSignal(sigBit%)
CALL FreeMem&(port&,38)

END SUB

'-----------------------------------------------------

SUB CreateIO(port&) STATIC

SHARED io&
opt&=2^0+2^16
io&=AllocMem&(62,opt&)
IF io&=0 THEN PRINT "Kein Speicherplatz!":EXIT SUB
POKE  io&+8,5
POKEL io&+14,port&
POKEW io&+18,12

END SUB

'-----------------------------------------------------
```

```
SUB DeleteIO(io&) STATIC

CALL FreeMem(io&,62)

END SUB
```

Hintergrund-Information zu Disketten-I/O:

Die obigen Beispiele sollen einen kleinen Einblick geben, wie man die trackdisk-device Befehle in BASIC-Programmen anwendet. Aber es liegt ganz an Ihnen, ob Sie es damit bewendet sein lassen wollen, oder ob Sie den vollen zur Verfügung stehenden Befehlssatz ausschöpfen. Hier finden Sie in einer Kurzbeschreibung, wie der I/O-Standard-Request-Block ausgefüllt werden muß, um mit allen Befehlen arbeiten zu können (näheres zum Standard Request Block siehe Kapitel I/O - Kommunikation mit der Außenwelt).

a) Datenmanipulations-Befehle

ETD READ und CMD READ

Diese beiden Kommandos transferieren Daten vom internen Track-Buffer zu einem beliebigen User-Buffer. Liegen die Daten für den verlangten Sektor bereits im Track-Buffer, so wird der Friede des Disk Drives nicht gestört. Andernfalls wird der Track, der den gesuchten Sektor enthält (1 Track = 22 Sektoren = 11 KByte), in den Track-Buffer geladen. Sollten die Daten im Track-Buffer zuvor verändert worden sein, so werden diese zunächst auf die Disk geschrieben, bevor der neue Track eingeladen wird. CMD READ überprüft nicht, ob die Diskette im Laufwerk gewechselt wurde, während ETD READ diese Tatsache pingelig überprüft.

CMD READ:

io Command	=	2
io Length	=	tdsector (=512) oder Vielfaches für mehr als einen Sektor
io Data	=	Anfangsadresse des User-Buffers (siehe "Memory Handling" für Schaffung eines User Buffers)
io Offset	=	gewünschter Sektor (Sektor*512)

ETD READ:

io Command: 32770, ansonsten wie CMD READ, jedoch zusätzlich, in einer Extended Request Form, die folgenden Felder:

iotd Count (io+48): Wechselzahl. Jedes Mal, wenn eine Diskette neu in ein Laufwerk eingeführt wird, erhöht sich der sogenannte "Disk Change Counter", dessen augenblicklichen Stand Sie durch das Kommando TD CHANGENUM erfahren können. Nur wenn diese Wechselzahl kleiner oder gleich der Zahl ist, die Sie in diesem Feld angegeben haben, kann die Anweisung ordnungsgemäß durchgeführt werden, andernfalls kommt es zu einem Error. Damit kann verhindert werden, daß während einer längerwährenden Diskettenoperation die Diskette hinterrücks ausgetauscht wird, etc. Sollten Sie EXTENDED DISK I/O wünschen, ohne den Disk-Change-Schnickschnack zu wollen, so können Sie in dieses Feld einen Wert wie 100000 einsetzen, denn es ist zweifelhaft, daß Sie die Diskette hunderttausend Mal wechseln.

iotd SecLabel (io&+52): Erlaubt den Zugriff auf die Sektor-Identifikations Köpfe. Jeder dieser Köpfe umfaßt 16 Bytes, die dem Disk Driver egal sind. Möchten Sie mit diesem Labels arbeiten, dann sollte dieses Feld auf einen Speicherbereich zeigen, der für jeden geladenen Sektor 16 Bytes bereit hält (Bei einem Track also 22*16 Bytes).

ETD WRITE und CMD WRITE

Diese beiden Befehle sind das Gegenstück zu den vorangegangenen beiden. Sie transferieren Daten aus dem user Buffer in den Track-Buffer, von wo die Daten auf Disk geschrieben werden. Mit diesen Befehlen können Sie also praktisch auf jedes einzelne Bit auf Diskette zugreifen.

CMD WRITE:

io Command:	3
io Length:	tdsector (=512) oder Vielfaches für weitere Sektoren
io Data:	Zeiger zu User Buffer
io Offset:	gewünschter Sektor (Sektor*512)

ETD WRITE:

io Command:	32771, ansonsten wie CMD WRITE, außerdem EXTENDED REQUEST, siehe EXTD READ.

ETD UPDATE und CMD UPDATE

Update sorgt dafür, daß der Inhalt des Track-Buffers auf Disk geschrieben wird. Dies geschieht normalerweise sowieso, aber wenn Sie es mal unbedingt selbst hervorrufen wollen, dann können Sie diesen Befehl verwenden. Der Track-Buffer wird allerdings nur dann auf Disk geschrieben, wenn in ihm Änderungen vorgenommen worden waren. ETD UPDATE prüft dabei wieder zunächst, ob die Diskette unberechtigterweise gewechselt wurde.

io Command: 4	CMD UPDATE
io Command: 32772	ETD UPDATE

ETD CLEAR und CMD CLEAR

Diese beiden Befehle erklären den Track-Buffer für ungültig. So muß beim nöchsten Zugriff zunächst wieder eine Leseoperation durchgeführt werden. Diese Befehle werden normalerweise benutzt, nachdem eine Diskette aus dem Laufwerk entfernt wurde, damit die alten Track-Daten nicht zufällig auf eine neue Diskette gelangen, wo sie sicherlich großen Schaden anrichten würden.

io Command: 5	CMD CLEAR
io Command: 32773	ETD CLEAR

ETD MOTOR und TD MOTOR

Mit diesem Befehl läßt sich der Motor eines Diskettenlaufwerkes kontrollieren. Er läßt sich ein- und ausschalten. Der vorangegangene Zustand des Motors wird dabei an den User zurückgegeben. Wird der Motor eingeschaltet, dann erlaubt der Disk Driver eine kleine Verzögerung, damit der Motor auf die benötigte konstante Geschwindigkeit kommt. Normalerweise ist es jedoch nicht nötig, den Disk Motor einzuschalten, da die entsprechenden Befehle READ und UPDATE das sowieso erledigen. Es ist allerdings Ihre Verantwortung, den Motor nach beendigter I/O-Tätigkeit wieder auszuschalten. Hier kommt dann dieser Befehl zum Zuge:

io Command: 9	TD MOTOR
io Command: 32777	ETD MOTOR
io Length:	1=Motor ein
	0=Motor aus

Nach Aufruf des Blocks enthält io Actual den alten Status des Motors.

TD FORMAT

TD FORMAT schreibt Daten auf eine noch nicht formatierte Diskette. Es wird auch dazu benutzt, Hard Errors zu überschreiben. TD FORMAT ignoriert sämtliche Track-Daten, die sich bereits auf der Disk befanden, und überprüft auch nicht, ob die Diskette gewechselt worden ist. Das io Data Feld muß auf mindestens einen Track Daten zeigen (mindestens 11 KByte). io Offset muß auf Track-Anfänge zeigen (track+*512*22), io Length muß Vielfache der Track-Länge enthalten (22*512). Nach Ablauf dieses Befehls sollten die auf Disk geschriebenen Daten durch einen READ-Lauf überprüft werden (VERIFY).

io Command: 11

alles weitere siehe CMD WRITE, die obigen Hinweise sind zu beachten!

TD REMOVE

Mit diesem Befehl läßt sich ein Software Interrupt ins System einbinden, der immer dann "zündet", wenn eine Diskette ins Laufwerk eingelegt oder herausgenommen wird.

io Command:	12
io Data:	Zeiger auf eine Software Interrupt Struktur

b) Status-Befehle

TD CHANGENUM

Dieser Befehl liefert den Wert des aktuellen Disk Change Counters (Wechselzähler) an den User zurück.

io Command: 13

TD CHANGESTATE

Mit diesem Befehl kann untersucht werden, ob sich in einem Diskdrive eine Diskette befindet. 0=Diskette im Drive, ansonsten keine drin.

io Command: 14

TD PROTSTATUS

Dieser Befehl liefert 0 wieder, wenn die Diskette nicht write-protected ist, der kleine Nippel also unten liegt.

io Command: 15

Die Ergebnisse dieser Funktionsaufrufe werden durch die Standard Request Struktur an den User zurückgeliefert und stehen im Feld "io Actual" zur Verfügung.

3. Programmieren unter C auf dem Amiga

Die Programmiersprache C ist im Kommen. Noch vor gar nicht langer Zeit nur unter Insidern im Gespräch, ist sie im Zuge des 68000-Prozessors inzwischen überall zu finden.

Auf dem Amiga läßt sich in C besonders effektiv programmieren, da ein großer Teil des Betriebsystemes selber in C geschrieben wurde und so alles wunderschön ineinander paßt.

Doch kurze Vorrede und lieber gleich rein in die C-Programmierung. Besorgen Sie sich einen C-Compiler (wir benutzen den Lattice-C-Compiler V3.3), und legen Sie los.

Die Listings der hier aufgeführten C-Programme haben wir mit Zeilennummern ausgedruckt, damit wir sie besser erklären können (indem wir auf bestimmte Zeilennummern hinweisen). Sie müssen beim Abtippen diese Zeilennummern natürlich weglassen, sonst meldet der C-Compiler eine ganze Menge Fehlermeldungen.

3.1 Beim Amiga verwendete Variablen-Typen

An die Kopfzeile jedes C-Programms gehört der Preprozessor-Befehl #include "exec/types.h". In diesem Headerfile werden einige Variablentypen definiert, die in allen anderen Headerfiles sowie auch in den meisten C-Programmen benutzt werden.

Als wichtigstes seien folgende Variablen genannt:

REGISTER: Auch sonst Register. Sollte öfter mal verwendet werden, das Programme dadurch um einiges schneller werden können.

LONG: vorzeichenbehafteter 32-Bit-Wert
ULONG: positiver 32-Bit-Wert
LONGBITS: Feld von 32 Bits, die einzeln manipuliert werden

WORD:	vorzeichenbehafteter 16-Bit-Wert
UWORD:	positiver 16-Bit-Wert
WORDBITS:	Feld von 16 Bits, die einzeln manipuliert werden
BYTE:	vorzeichenbehafter 8-Bit-Wert
UBYTE:	positiver 8-Bit-Wert
BYTEBITS:	Feld von 8 Bits, die einzeln manipuliert werden
STRPTR:	Zeiger auf String
APTR:	Absoluter Zeiger in den Speicher
FLOAT:	normal float
DOUBLE:	normal double
COUNT:	Zähler im Bereich von -32768 bis 32767
UCOUNT:	Zähler im Bereich von 0 bis 65535
BOOL:	Flag

Zusätzlich existieren noch folgende Makros:

TRUE: 1
FALSE: 0

NULL:	0, sollte verwendet werden um zu kennzeichnen, daß diese Variable nicht benutzt wird
BYTEMASK:	0xff
FOREVER:	zu finden im Headerfile intuition/intuition.h. Entspricht for (;;)
NOT:	gleich "!", jedoch um einiges verständlicher; auch aus dem Headerfile intuition/intuition.h

3.2 AmigaDOS

Neben Intuition, der grafik-orientierten Benutzerschnitt- stelle, verfügt der Amiga auch noch über eine "normale" Schnittstelle - AmigaDOS. AmigaDOS erinnert an Betriebssysteme wie MS-DOS oder CP/M. Allerdings gibt es auch recht erhebliche Unterschiede. So ist AmigaDOS z.B. multitasking-fähig.

Benutzt wird AmigaDOS wahrscheinlich nicht gerade von Anwendern - diese sind mit Intuition besser und einfacher bedient. Als Programmierer wird man jedoch nicht umhin kommen, AmigaDOS zu benutzen.

Dies als kurze Einleitung. Mehr Informationen erhalten Sie im AmigaDOS-Manual von Commodore.

Nein, das Kapitel ist damit noch nicht zu Ende. Hier finden Sie noch einige Informationen, die nicht im Manual zu finden sind. Außerdem folgen natürlich auch noch einige kurze Listings:

3.2.1 AmigaDOS-Strukturen

Wie der restliche Amiga greift auch AmigaDOS nicht auf festgelegte Speicherstellen zurück (vielleicht erinnern Sie sich noch an die Poke-Listings bei früheren Rechnern, z.B. beim C 64?). Auch AmigaDOS benutzt Strukturen. Indem man auf diese zurückgreift, kann man viele interessante Informationen erhalten und noch mehr interessante Aktionen hervorrufen.

Hier erklären wir kurz die von AmigaDOS benutzten Strukturen - sofern wir über sie Bescheid wissen. Anwendungsbeispiele finden Sie im nächsten Teil dieses Kapitels.

Die Grundstruktur beim AmigaDOS ist die Struktur "DosLibrary". Sie erhalten diese durch folgenden Aufruf:

```
doslibrary = (struct DosLibrary *)
                    OpenLibrary ("dos.library", 0);
```

Die Struktur DosLibrary setzt sich aus folgenden Komponenten zusammen:

```
struct Library dl_lib
APTR dl_Root
APTR dl_GV
LONG dl_A2
LONG dl_A5
LONG dl_A6
```

Die Komponenten dl_A2, dl_A5 und dl_A6 stellen Registerinhalte da. Diese Komponenten werden von AmigaDOS zum Zwischenspeichern verwendet und sollten nicht verändert werden. Die Komponente dl_GV ist ein Pointer zu einer Sprungtabelle, genannt "Shared Global Vector". Diese Sprungtabelle wird intern von AmigaDOS benutzt. Ein sinnvoller Zugriff auf diese Komponente setzt recht gute Kenntnisse des AmigaDOS voraus. Da wir diese zur Zeit noch nicht haben, können wir auch nicht näher auf diese Adresse eingehen. Wer etwas darüber herausfindet: Bitte unbedingt schreiben!

Mit dl_lib kann man auf die Library-Struktur der AmigaDOS-Library zugreifen. Diese Struktur enthält die Versions-Nummer, die Knoten-Struktur und ähnliches. Diese Struktur wird an anderer Stelle in diesem Buch beschrieben.

So richtig interessant wird's wieder bei dl_Root. Diese Komponente ist ein Zeiger auf eine RootNode-Struktur. In dieser Struktur finden Sie folgende Informationen:

```
BPTR rn_TaskArray
BPTR rn_ConsoleSegment
struct DateStamp rn_Time
LONG rn_RestartSeg
BPTR rn_Info
```

Hier enthält rn_TaskArray[0] die maximale Nummer der CLIs, und rn_TaskArray[1] - rn_TaskArray[n] die Message-Ports für die einzelnen CLIs 1 - n. Die Komponente "rn_ConsoleSegment" enthält die Adresse der Seglist für die CLI.

In "rn_Time" ist die momentane Zeit abgespeichert. Die Struktur DateStamp hat übrigens folgende Komponenten:

```
LONG ds_Days
LONG ds_Minute
LONG ds_Tick
```

"ds_Days" enthält die Anzahl der Tage, die seit dem 1. Januar 1978 vergangen sind, "ds_Minute" enthält die Anzahl der Minuten seit Mitternacht, und "ds_Tick" enthält die Anzahl der Ticks seit Anfang der letzten Minute.

Weiter zur RootNode-Struktur: In "rn_RestartSeg" finden Sie die SegList für den Disk-Validator-Prozeß. Dieser Prozeß wird jedesmal in Gang geworfen, wenn eine Diskette in ein Laufwerk eingelegt wird. Hier könnten Sie einsetzen, wenn Sie ein bestimmtes Programm jedesmal ausführen wollen, wenn eine neue Diskette eingelegt wird.

Mit "rn_Info" kann schließlich auf eine DosInfo-Struktur zugegriffen werden. Diese Struktur wird momentan fast gar nicht benutzt, und deshalb hier auch nicht weiter aufgeführt. Sie ist für spätere Erweiterungen (z.B. in Richtung Netzwerk) gedacht.

Eine weitere wichtige Struktur ist FileInfoBlock. Diese Struktur wird von den AmigaDOS-Befehlen "Examine" und "ExNext" benutzt. Sie muß auf einer 4-Byte-Grenze liegen, und sollte so mittels der Exec-Funktion "AllocMem" definiert werden (AllocMem läßt Speicher-Blöcke immer auf einer 8-Byte-Grenze anfangen).

Im FileInfoBlock findet sich folgendes:

```
LONG fib_DiskKey
LONG fib_DirEntryType
char fib_FileName[108]
LONG fib_Protection
LONG fib_EntryType
LONG fib_Size
```

```
LONG fib_NumBlocks
struct DateStamp fib_Date
char fib_Comment[116]
```

Wenn fib_DirEntryType kleiner als 0 ist, handelt es sich um ein File. Ist fib_DirEntryType jedoch größer als 0, so haben Sie ein Directory vor sich.

In fib_FileName finden Sie den Namen des Files (abgeschlossen mit einer 0). Zur Zeit werden jedoch nicht alle 108 Zeichen benutzt, sondern nur 30.

fib_Protection gibt an, ob und wenn ja das File geschützt ist. Bis jetzt haben nur die ersten vier Bits eine Bedeutung, und zwar folgende:

Bit 0: delete
Bit 1: execute
Bit 2: write
Bit 3: read

Die Größe des Files finden Sie in fib_Size, und die Anzahl der belegten Blöcke in fib_NumBlocks.

fib_Date gibt die Zeit an, zu der das File zum letzten mal geändert wurde.

Und schließlich steht in fib_Comment[116] noch der Kommentar zu dem File (wieder mit 0 abgeschlossen).

Und nun die letzte Struktur: InfoData. Diese Struktur wird von der AmigaDOS-Funktion "Info" benutzt. Sie gibt Informationen über die Diskette aus:

```
LONG id_NumSoftErrors
LONG id_UnitNumber
LONG id_DiskState
LONG id_NumBlocks
LONG id_NumBlocksUsed
LONG id_BytesPerBlock
```

```
LONG id_DiskType
BPTR id_VolumeNode
LONG id_InUse
```

Die meisten Komponenten dürften selbsterklärend sein. Falls das Flag id_InUse gleich 0 ist, wird die Diskette nicht benutzt. Zahlreiche Makros zur InfoData-Struktur finden Sie in dem Header-File "libraries/dos.h".

3.2.2 Programme mit AmigaDOS-Funktionen und Strukturen

Hier finden Sie einige kleine Routinen, die mit den AmigaDOS-Funktionen sowie den AmigaDOS-Strukturen arbeiten. Zu den Routinen haben wir meistens noch eine Hauptfunktion aufgeführt, um die Anwendung der jeweiligen Routine zu zeigen.

3.2.2.1 Anzahl der freien Blöcke auf Disk bestimmen

Mit Hilfe dieser Routine können Sie die Anzahl der freien Blöcke auf einer Diskette bestimmen. Wir haben diese Routine NumFreeBlocks genannt, und sie hat folgenden Code:

```
#include <libraries/dosextens.h>
#include <exec/memory.h>


LONG NumFreeBlocks (name)
   char name[];
{
   BPTR lock;
   struct InfoData *infodata;
   LONG numfreeblocks;

   infodata = AllocMem (sizeof (struct InfoData), MEMF_CLEAR
  );
```

```
    lock = Lock (name, ACCESS_READ);
    if (lock == NULL) return (-1);

    if (Info (lock, infodata) == FALSE) return (-1);


    numfreeblocks = (infodata->id_NumBlocks) - (infodata->id_
   NumBlocksUsed);

    UnLock (lock);

    return (numfreeblocks);
}


main()
{
    LONG numfreeblocks;


    numfreeblocks = NumFreeBlocks ("df1:");
    if (numfreeblocks == -1)
    {

       printf ("Hey, ich hab Probleme!!!\n");
       exit (FALSE);
    }


    printf ("Freie Bloecke von df1: %d\n", numfreeblocks);
}
```

Einige Worte zur Erklärung: Zuerst einmal werden die notwendigen Headerfiles eingebunden. Mit "dosextens.h" werden gleichzeitig auch noch so wichtige Headerfiles wie "exec/types.h" und "libraries/dos.h" in den Programmtext eingefügt. Das File "exec/memory.h" muß auch eingefügt werden, da dort die für die AllocMem-Funktion notwendigen Makros definiert sind.

Und dann beginnt auch schon die Funktion. Aufgerufen wird die Funktion NumFreeBlocks mit dem Argument "name". Diese Variable enthält den gewünschten Drive-Namen. Dies könnte, wie in unserem Programm, "df1:" sein; es sind jedoch auch Namen wie "Workbench:" oder "ram:" erlaubt.

Sodann werden die benötigten Variablen definiert. Der Zeiger auf eine Struktur vom Typ "InfoData" muß jedoch auf einer 4-Byte-Grenze beginnen. Dieses erreicht man durch Verwendung der exec-Funktion "AllocMem". Das Attribut "MEMF_CLEAR" sorgt dafür, daß der gewünschte Speicherbereich mit Nullen gefüllt wird. Dann wird mit der AmigaDOS-Funktion "Lock" ein Zeiger auf die FileLock-Struktur zu der gewünschten Disk bestimmt. Falls dieser Zeiger gleich Null ist, hat irgendetwas nicht ganz geklappt, und NumFreeBlocks springt vorsichtshalber mit dem Fehler-Code -1 zurück.

Hat alles geklappt, so wird mit der AmigaDOS-Funktion "Info" die InfoData-Struktur mit den Werten der gewünschten Disk gefüllt. Auch hier wird bei einem auftretenden Fehler wieder mit -1 zurückgesprungen. Schließlich wird die Anzahl der freien Blöcke bestimmt, indem von der Anzahl der möglichen Blöcke die Anzahl der schon belegten Blöcke abgezogen wird. Danach muß der Lock wieder freigegeben werden. Dies geschieht mittels der AmigaDOS-Funktion UnLock. In main wird als Beispiel die Funktion NumFreeBlocks mit dem Parameter "df1:" aufgerufen.

3.2.2.2 Größe eines Files bestimmen

Diese Funktion bestimmt die Größte eines beliebigen File auf Disk:

```
#include <libraries/dosextens.h>
#include <exec/memory.h>
```

```
LONG FileSize (name)
   char name[];
{
   BPTR lock;
   struct FileInfoBlock *fileinfoblock;


   fileinfoblock = AllocMem (sizeof (struct FileInfoBlock),
  MEMF_CLEAR);


   lock = Lock (name, ACCESS_READ);
   if (lock == NULL) return (-1);


   if (Examine (lock, fileinfoblock) == FALSE) return (-1);

   UnLock (lock);

   return (fileinfoblock->fib_Size);
}



main()
{
   LONG filesize;


   filesize = FileSize ("df0:preferences");
   if (filesize == -1)
   {
      printf ("Nanu, wo hamwa denn das File...\n");
      exit (FALSE);
   }

   printf ("Groesse von preferences: %d\n", filesize);
}
```

Diese Funktion ist der obigen ziemlich ähnlich. Zuerst werden wieder die benötigten Header-Files in den Text eingebunden. Innerhalb der Funktion FileSize werden wieder zuerst die benötigten Variablen definiert, wobei auch die Struktur vom Typ FileInfoBlock auf einer 4-Byte-Grenze beginnen muß. Sodann wird der lock zu dem gewünschten File besorgt; anschließend wird mit der AmigaDOS-Funktion Examine der FileInfoBlock zu dem gewünschten File in den Speicher geholt. Ist hierbei ein Fehler aufgetreten, wird FileSize wiederum mit -1 verlassen. Auch hier muß danach der Lock wieder freigegeben. Ist alles gut gegangen, springt FileSize mit der File-Größe in Bytes wieder zurück.

In main wird die Funktion ausprobiert; als File wird df1:Preferences genommen. Ist ein Fehler aufgetreten, d.h., wurde FileSize mit dem Wert -1 verlassen, so gibt der Amiga eine entsprechende Meldung aus.

3.2.2.3 File vorhanden?

Mit Hilfe dieser kleinen Routine können Sie testen, ob ein bestimmtes File (oder auch Directory) vorhanden ist. Die Funktion heißt FileExists und wird folgendermaßen aufgerufen:

```
FileExists (Name in Gänsefüßchen)
```

Als Ergebnis liefert die Funktion einen Boolean-Wert zurück - TRUE, wenn das File existiert, und FALSE, falls nicht. Hier das Listing:

```
#include <libraries/dosextens.h>

BOOL FileExists (name)
   char name[];
{
   BOOL fileexists;
```

```
   fileexists = Lock (name, ACCESS_READ);
   UnLock (fileexists);

   if (fileexists == NULL)
      return (FALSE);
   else return (TRUE);
}


main ()
{
   if ((FileExists ("df0:Clock")) == FALSE)
      printf ("Oh, keine Uhr in Drive df0:\n");
   else printf ("Da isse ja, die kleine Uhr.\n");


   if ((FileExists ("df1:Upplands_Vasby")) == FALSE)
      printf ("As far as my eyes can see - kein solches File
   in Sicht\n");
   else printf ("Welch Zufall! Schwedenfahrer?!\n");
}
```

Die Routine beruht darauf, daß die AmigaDOS-Funktion Lock den Wert 0 zurückgibt, falls das gewünschte File nicht existiert. Natürlich hätte man diesen Test auch mittels der AmigaDOS-Funktion Open durchführen können - Lock hat jedoch viel weniger "overhead" (siehe Sachworterklärung im Anhang) als Open.

3.2.2.4 File-Kommentar lesen

Diese Funktion stellt das Gegenstück zu der AmigaDOS-Funktion SetComment da. Wir haben sie sinnvollerweise GetComment genannt, und sie wird so aufgerufen:

```
GetComment (Name des Files in Gänsefüßchen)
```

Als Ergebnis erhalten Sie einen Zeiger auf dèn Kommentar zurück. Hier nun das Listing:

```
#include <libraries/dosextens.h>
#include <exec/memory.h>


char *GetComment (name)
   char name[];
{
   BPTR lock;
   struct FileInfoBlock *fileinfoblock;


   fileinfoblock = AllocMem (sizeof (struct FileInfoBlock),
  MEMF_CLEAR);


   lock = Lock (name, ACCESS_READ);
   if (lock == NULL) return ((char *) -1);


   if (Examine (lock, fileinfoblock) == FALSE) return ((char
   *) -1);
```

```
   UnLock (lock);

   return ((char *) &(fileinfoblock->fib_Comment));
}


main ()
{
   char *comment;

   comment = GetComment ("df0:Preferences");
   if (comment == (char *) -1)
   {
      printf ("Leider keine Preferences in df0:\n");
      exit (FALSE);
   }

   printf ("Der Kommentar zu Preferences lautet:\n%s\n", com
  ment);
}
```

In den Zeilen 17, 20, 26 sowie 37 muß eine Cast-Operation verwendet werden, damit der zurückgegebende bzw. verglichene Wert vom geforderten Typ ist. Sollten Ihre Preferences keinen Kommentar besitzen, so sollten Sie zum Testen der Funktion GetComment einen hinzufügen. Dies können Sie zum Beispiel von der Workbench aus mit dem Menüpunkt Info erreichen. Ansonsten ist die Funktion den oberen dreien (bes. der Funktion FileSize) ziemlich ähnlich und braucht daher nicht weiter erklärt zu werden.

3.2.2.5 GetProtection

Diese Funktion ist das Gegenstück zu der AmigaDOS-Funktion SetProtection. Wird diese Funktion mit einem File-Namen als Argument aufgerufen, so gibt sie die Protection-Bits wieder zurück. Diese Bits können Sie dann mit Hilfe der Makros in dem Headerfile "libraries/dos.h" entschlüsseln. Das Listing der Routine:

```
#include <libraries/dosextens.h>
#include <exec/memory.h>


LONG GetProtection (name)
   char name[];
{
   BPTR lock;
   struct FileInfoBlock *fileinfoblock;


   fileinfoblock = AllocMem (sizeof (struct FileInfoBlock),
  MEMF_CLEAR);


   lock = Lock (name, ACCESS_READ);
   if (lock == NULL) return (-1);


   if (Examine (lock, fileinfoblock) == FALSE) return (-1);


   UnLock (lock);


   return (fileinfoblock->fib_Protection);
}


main()
{
   LONG protection;


   protection = GetProtection ("df0:Preferences");
   if (protection == -1)
   {
      printf ("?FILE NOT FOUND ERROR\nREADY.\n");
```

```
      exit (FALSE);
   }

   if (protection && FIBF_DELETE)
      printf ("Delete-Protection off\n");
   else printf ("Delete-Protection on\n");
}
```

Da zur Zeit nur das Delete-Bit bei den Protection-Bits benutzt wird, testet das Programm auch nur dieses Bit ab.

3.2.2.6 Effektivere Warteschleifen mit Delay

Manchmal ist der Computer zu langsam, manchmal zu schnell. Ist er zu langsam, muß man sich einen neuen kaufen. Ist er jedoch zu schnell, so packt man einfach eine Warteschleife ins Programm. Diese ist in C schnell geschrieben, und sieht meistens ungefähr so aus:

```
for (t = 0; t < 100000; t++) ;
```

Doch der Amiga ist ein Multitasking-Computer. Und aus diesem Grund wäre obrige Anweisung sehr uneffektiv. Denn der Amiga würde die for-Anweisung wie jede andere Anweisung auch ausführen, d.h. der Task würde genau sein Quentchen Zeit in Anspruch nehmen, obwohl er in dieser Zeit eigentlich nur wartet. Folgendes Programm verdeutlicht das:

```
#include <exec/types.h>
#include <exec/tasks.h>
```

```
main ()
{
   struct Task *task;
   UCOUNT zaehler;


   task = FindTask (0);

   SetTaskPri (task, 30);

   for (zaehler = 1; zaehler < 1000; zaehler++)
      printf ("Zum %d. mal: Multitasking adieu...\n", zaehle
  r);
}
```

Tippen Sie das Programm ab, compilieren und linken Sie es. Nun starten Sie ein beliebiges Programm, zum Beispiel Graphicraft, mit "run Graphicraft". Malen Sie nun ein bißchen herum. Holen Sie anschließend mit "LeftAmiga + n" den Workbench-Screen nach vorne, und starten Sie das compilierte Programm. Holen Sie nun mit der Tastenkombination "LeftAmiga + m" den Graphicraft-Screen wieder nach vorne, und versuchen Sie, wieder ein bißchen zu malen. Leider geht das nicht mehr.

Intuition bekommt noch etwas System-Zeit, so daß Menus, Screens, Windows und dergleichen bedient werden können. Andere Tasks erhalten jedoch nichts mehr.

Zur Erklärung: Das Programm setzt zuerst einmal seine eigene Task-Priorität hoch. Dazu holt es sich erst mittels der exec-Funktion FindTask einen Zeiger auf die Task-Struktur. In Zeile 13 wird sodenn die Taskpriorität durch die exec-Funktion SetTaskPri auf 120 gesetzt. Danach wird ziemlich lange gewartet. Damit Sie sicher sind, daß das Programm auch schön läuft, wird jedesmal eine Meldung ausgegeben. Sollten Sie auch noch die printf-Anweisung herausnehmen, so wird selbst Intuition nicht mehr funktionieren (wahrscheinlich hat die Grafik-Routine, mit deren Hilfe Zeichen ausgegeben werden, eine niedrigere Priorität als Intuition; wird nichts mehr ausgegeben, kommt selbst Intuition nicht mehr dazwischen). Probieren Sie nun ein weiteres Programm:

```
#include <exec/types.h>
#include <exec/tasks.h>


main ()
{
   struct Task *task;


   task = FindTask (0);

   SetTaskPri (task, 30);


   Delay (3000);
}
```

Dieses Programm ist dem obigen ziemlich ähnlich. Statt der Warteschleife mit for wurde der AmigaDOS-Befehl Delay eingesetzt. Dieser Befehl erwartet als Argument die Anzahl von Ticks, die abgewartet werden sollen (eine Sekunde besteht aus ca. 50 Ticks). Die Besonderheit von Delay bemerken Sie jedoch, wenn Sie die gleiche Prozedur wie oben beschrieben durchführen. Diesmal läßt sich noch mit Graphicraft arbeiten, wenn das Programm gestartet ist. Delay setzt den Task nämlich in den sog. Wait-Modus; solange der Task auf die Antwort des Timer-Devices wartet, können andere Tasks ablaufen, und Multitasking ist gewährleistet.

Also: Beim Warten immer Delay verwenden, da sonst andere Tasks keine System-Zeit bekommen.

3.3 Grafik-Programmierung auf dem Amiga

Vorbei sind die Zeiten, wo der Computer geradeso Schrift darstellen konnte. Vorbei sind Auflösungen von 176 * 184 Punkten. Ein moderner Computer hat 640 * 400 Auflösung, jede Menge

Farben, eingebaute Grafikbefehle. Der Amiga hat noch einiges mehr: Screens, Windows, Menüs, einfache Rasterinterruptprogrammierung, extra Grafik-Prozeßor, Requesters, Gadgets, und, und, und.

Wie Sie alles dies benutzen können, wollen wir Ihnen in diesem Kapitel zeigen.

3.3.1 Screens und Windows

Die Basis für jedwede Grafik-Darstellung ist der Screen. Das war auch beim VC-20 und C64 so. Zum Unterschied zu diesen beiden und auch den meisten anderen Computern kann der Amiga jedoch nicht nur einen, sondern beliebig viele Screens haben. Dabei kann jeder Screen völlig andere grafische Eigenschaften besitzen. Ein Screen könnte zum Beispiel eine Auflösung von 640 * 400 Punkten bei acht Farbregistern besitzen, während gleichzeitig ein anderer Screen eine Auflösung von 320 * 200 Punkten mit zwei Farbregistern hat. Doch nicht nur Auflösung und Tiefe können bei einem Screen festgelegt werden. So kann zum Beispiel auch für jeden Screen bestimmt werden, was für ein Zeichensatz verwendet wird, wo er liegt, woher er seine BitMap bekommt.

Das zweite wichtige Element sind Windows. Diese werden innerhalb eines bestimmten Screens eröffnet und können diesen auch nicht verlassen. Windows können in beliebiger Anzahl eröffnet werden - und auch sie können völlig verschiedenes Aussehen haben. Bei Windows kann man unter anderem Kriterien wie Größe, Lage, Titel, Gadgets und verwendete Flags festlegen.

Screens und Windows werden von Intuition verwaltet. Sie rufen bestimmte Routinen auf, übergeben die gewünschten Eigenschaften Ihrer Screens & Windows und haben damit Ruhe.

Wenn ein Window vom Benutzer vergrößert oder verkleinert wird, wenn ein Window hin- und hergeschoben wird, wenn ein Screen nach vorne oder hinten geholt wird - Intuition erledigt alles.

Kommen wir zu den Einzelheiten. Dies am besten anhand eines Beispiels:

```
#include <exec/types.h>
#include <intuition/intuition.h>

struct IntuitionBase *IntuitionBase;

main()
{
   struct Screen *Screen;
   struct Window *Window;
   struct NewScreen NewScreen;
   struct NewWindow NewWindow;

   IntuitionBase = (struct IntuitionBase *)
                         OpenLibrary ("intuition.library", 0)
  ;
   if (IntuitionBase == NULL)
   {
      printf ("Tschuldigung, keine Intuition-Library\n");
      exit (FALSE);
   }

   NewScreen.LeftEdge = 0; NewScreen.TopEdge = 0;
   NewScreen.Width = 640; NewScreen.Height = 200;
   NewScreen.Depth = 2;
   NewScreen.DetailPen = 0; NewScreen.BlockPen = 1;
   NewScreen.ViewModes = HIRES;
   NewScreen.Type = CUSTOMSCREEN;
   NewScreen.Font = NULL;
   NewScreen.DefaultTitle = "Ein neuer Screen";
   NewScreen.Gadgets = NULL;
   NewScreen.CustomBitMap = NULL;

   Screen = (struct Screen *) OpenScreen (&NewScreen);
   if (Screen == NULL)
   {
      printf ("Screen laesst sich nicht oeffnen!!!\n");
```

```
    exit (FALSE);
 }


 NewWindow.LeftEdge = 20;
 NewWindow.TopEdge = 20;
 NewWindow.Width = 400;
 NewWindow.Height = 100;
 NewWindow.DetailPen = 0;
 NewWindow.BlockPen = 1;
 NewWindow.IDCMPFlags = CLOSEWINDOW;
 NewWindow.Flags = WINDOWSIZING | WINDOWDEPTH | WINDOWCLOS
E | WINDOWDRAG |
                  SMART_REFRESH | ACTIVATE | NOCAREREFRES
H;
 NewWindow.FirstGadget = NULL;
 NewWindow.CheckMark = NULL;
 NewWindow.Title = "*** Dies ist ein Test-Window ***";
 NewWindow.Screen = Screen;
 NewWindow.BitMap = NULL;
 NewWindow.MinWidth = 400; NewWindow.MinHeight = 20;
 NewWindow.MaxWidth = 640; NewWindow.MaxHeight = 200;
 NewWindow.Type = CUSTOMSCREEN;

 Window = (struct Window *) OpenWindow (&NewWindow);
 if (Window == NULL)
 {
    printf ("Probleme mit dem Window...\n");
    exit (FALSE);
 }
```

```
    Wait (1 << Window->UserPort->mp_SigBit);


    CloseWindow (Window);
    CloseScreen (Screen);
    CloseLibrary (IntuitionBase);

}
```

Tippen Sie das Programm ab, compilieren und linken Sie es, und starten Sie es dann.

Das Window können Sie nun wie jedes andere Window bearbeiten. Sie können es vergrößern, verkleinern, verschieben und schließen. Wenn sie es schließen, so wird das Programm beendet. Auch der Screen läßt sich ganz normal verwenden. Mit der Menüleiste können Sie ihn verschieben, und mit LeftAmiga + m nach vorne bringen. So, genug ausprobiert? Dann zu den Erklärungen:

In den Zeilen 1 und 2 werden die notwendigen Header-Files in den Programmtext eingebunden. In intuition/intuition.h sind alle Strukturen, die etwas mit Intuition zu tun haben, zu finden (zum Beispiel alle Strukturen zu Screens und Windows, jedoch auch zu Gadgets, Requesters und dergleichen). Danach wird ein Zeiger auf eine IntuitionBase-Struktur definiert. Dieser Zeiger wird gebraucht, wenn die Intuition-Library eröffnet wird. Und dann beginnt auch schon main.

Hier werden zuerst alle benötigten Variablen definiert. Dann wird die Intuition-Library geöffnet. Dies geschieht mit dem exec-Befehl OpenLibrary. Das zweite Argument zu diesem Befehl gibt die gewünschte Versionsnummer an. In den Zeilen 25 - 34 werden die einzelnen Komponenten einer NewScreen-Struktur beschrieben. Diese NewScreen-Struktur beschreibt den Screen. Die einzelnen Komponenten haben hierbei folgende Bedeutung:

LeftEdge: Gibt den X-Wert des Anfangspunktes des Screens an. Diese Komponente wird in der jetzigen Ausgabe von Intuition

nicht benutzt. Damit Ihr Programm jedoch auch unter späteren Versionen von Intuition läuft, sollten Sie diese Komponente auf 0 setzen.

TopEdge: Y-Wert des Anfangspunktes. In unserem Beispiel ist der Anfangspunkt des Screens in der linken oberen Ecke des Bildschirmes.

Width: Gewünschte Weite des Screens in Pixels.

Height: Gewünschte Höhe des Screens - auch in Pixels.

Depth: Anzahl der Bitmaps. Daraus ergibt sich die Anzahl der verfügbaren Farbregister. In unserem Beispiel sind es 2 Bitmaps, also 2^2 = 4 Farben.

DetailPen: Nummer des Farbregisters, aus dem die Farbe für einige Sachen wie den Title-Text oder Gadgets genommen werden soll.

BlockPen: Wie oben, jedoch für block fills, zum Beispiel Title-Hintergrund.

ViewModes: Hier werden einige Flags gesetzt. Bis jetzt sind folgende verfügbar:

HIRES: Muß gesetzt werden, wenn High-Resolution gefragt ist (wie in unserem Beispiel).

INTERLACE: Schaltet Interlace-Modus für diesen Screen ein. Sollten Sie dieses Flag verwenden, und der Compiler gibt bekannt, daß das Flag gar nicht bekannt ist, so schauen Sie sich mal das Includefile graphic/view.h an und ändern Sie ggf. den Macro LACE in INTERLACE um.

SPRITES: Muß gesetzt werden, wenn Sprites verwendet werden sollen.

DUALPF: Setzen Sie dieses Flag, wenn Sie zwei Playfields haben wollen.

HAM: Schaltet Hold-and-Modify-Modus ein.

Types: Hier können folgende Flags gesetzt werden:

CUSTOMSCREEN: Sollte auf jeden Fall gesetzt werden.

CUSTOMBITMAP: Setzen Sie dieses Flag, wenn Sie eine eigene Bitmap für diesen Screen haben wollen.

Font: Ein Zeiger auf eine TextAttr-Struktur. Diese Struktur bestimmt den Zeichensatz, in dem alle von Intuition ausgegebenen Texte erscheinen sollen (zum Beispiel Title, Menüs). In unserem Beispiel wurde diese Komponente auf Null gesetzt - dann nimmt Intuition den Standard-Zeichensatz.

DefaultTitle: Zeiger auf gewünschten Title. Setzen Sie diesen Zeiger auf Null, wenn Sie keinen Titel haben wollen.

Gadgets: Wird zur Zeit nicht benutzt und sollte auf Null gesetzt werden.

CustomBitMap: Muß einen Zeiger auf eine BitMap-Struktur enthalten, wenn Sie für den Screen eine eigene Bitmap verwenden wollen. Denken Sie auch daran, daß CUSTOMBITMAP-Flag in der Komponente Types zu setzen.

Nachdem alle gewünschten Werte in der NewScreen-Struktur abgespeichert wurden, wird der Screen eröffnet. Dazu wird die Intuition-Funktion OpenScreen aufgerufen. Diese Funktion benötigt als Argument die Adresse einer NewScreen-Struktur. Ist ein Fehler aufgetreten, so gibt diese Funktion den Wert Null zurück. In diesem Fall gibt das Programm eine entsprechende Meldung aus, und verabschiedet sich.

Ist alles gut abgelaufen, gibt OpenScreen einen Zeiger auf eine Screen-Struktur zurück. Diese Struktur ist im Headerfile intuition/intuition.h definiert und enthält folgende Komponenten:

NextScreen: Dies ist ein Zeiger auf eine weitere Screen-Struktur. Alle offenen Screens sind über diese Zeiger in einer Liste miteinander verkettet.

FirstWindow: Diese Komponente ist ein Zeiger auf die Window-Struktur des ersten Windows in dem Screen.

LeftEdge, TopEdge, Width, Height: wie bei NewScreen

MouseY, MouseX: Hier sind die Koordinaten der Mouse abgelegt.

Flags: Diese Komponente kann folgende Flags enthalten:

WBENCHSCREEN:	Workbench-Screen
CUSTOMSCREEN:	Eigener Screen
SHOWTITLE:	Wird gesetzt bei Aufruf von ShowTitle
BEEPING:	Wird gesetzt wenn Screen blinkt.
CUSTOMBITMAP:	Ist gesetzt, wenn der Programmierer eine eigene Bitmap wollte.

Title: Titel des Screens.

DefaultTitle: Title für Windows ohne Screen-Title.

BarHeight, BarVBorder, BarHBorder, MenuVBorder, MenuHBorder, WBorTop, WBorLeft, WBorRight, WBorBottom: Diese Variablen geben die Größe der Title-Leiste für diesen Screen und alle Windows innerhalb des Screens an.

Font: Zeiger auf eine TextAttr-Struktur. Diese Struktur gibt den Zeichensatz an, der für alle von Intuition ausgegebenen Texte innerhalb dieses Screens gelten soll.

ViewPort: ViewPort-Struktur für diesen Screen. In der ViewPort-Struktur werden die Eigenschaften des Screens für den Copper verständlich dargestellt.

RastPort: RastPort-Struktur für diesen Screen.

BitMap: BitMap-Struktur für den Screen. Legt das Aussehen der Bitmap fest.

LayerInfo: Layer_Info-Struktur. Diese Komponente sowie ViewPort, RastPort und BitMap sind für Programmierer gedacht, die direkt in den Grafik-Speicher schreiben wollen. Da diese niedrige Ebene der Grafik-Programmierung ziemlich schwierig und meistens auch nicht unbedingt notwendig ist, wird auf diese vier Variablen nicht näher eingegangen.

FirstGadget: Zeiger auf die Gadget-Struktur des ersten Gadgets für diesen Screen. Alle Gadgets werden in einer Liste gespeichert; nicht in dieser Liste sind die System-Gadgets.

DetailPen, BlockPen: wie bei NewScreen.

SaveColor0: System-Variable für DisplayBeep.

BarLayer: Zeiger auf die Layer-Struktur der Titel-Leiste für diesen Screen.

ExtData: ???

UserData: Zeiger auf irgendwelche Daten des Programmierers.

Nachdem jetzt der Screen geöffnet wurde, ist es Zeit, ein Window in diesem Screen zu eröffnen. Dazu wird in den Zeilen 44-60 eine NewWindow-Struktur beschrieben. Diese Struktur beschreibt die Eigenschaften des Windows, und besitzt folgende Komponenten:

LeftEdge, TopEdge, Width, Height: Beschreiben Größe und Lage des Fensters (siehe NewScreen).

DetailPen, Blockpen: siehe NewScreen

IDCMPFlags: Hier geben Sie die Fälle an, für die Intuition Ihnen ein Signal via dem IDCMPort übergeben soll. Folgende Flags sind dabei möglich:

MOUSEBUTTON: Muß gesetzt werden, wenn Intuition Ihnen ein Signal senden soll, wenn der Benutzer eine der beiden Maustasten gedrückt hat.

MOUSEMOVE: Wenn dieses Flag gesetzt ist, werden Mausbewegungen berichtet.

DELTAMOVE: Ist DELTAMOVE gesetzt, werden die Mausbewegungen nicht als absolute Werte, sondern als Delta-Werte (Anzahl der Änderungen bezüglich der letzten Position) angegeben.

GADGETDOWN: Hier wird eine Nachricht ausgegeben, wenn der Benutzer ein Gadget aussucht, daß mit gesetztem GADGIMMEDIATE-Flag erstellt wurde.

GADGETUP: Wie oben, jedoch für Gadgets, die mit RELVERIFY erstellt wurden.

CLOSEWINDOW: Intuition sendet Ihnen eine Nachricht wenn das CloseWindow-Gadget ausgesucht wurde (man/frau beachte: Intuition schließt Ihr Window nicht; es sendet nur eine Nachricht!).

NEWSIZE: Bei diesem Flag gesetzt erhalten Sie eine Nachricht, wenn der Benutzer die Größe des Windows verändert hat.

REFRESHWINDOW: Hier erhält Ihr Programm eine Nachricht, wenn Intuition der Meinung ist, daß Ihr Window wieder erneuert werden muß. Dieses Flag ergibt nur Sinn für Windows vom Typ SIMPLE_REFRESH und SMART_REFRESH.

NEWPREFS: Ihr Programm erhält von Intuition eine Nachricht, wenn die Preferences vom Benutzer geändert wurden. Anbei: Jedes Programm, daß dieses Flag gesetzt hat, bekommt eine Nachricht - nicht nur das aktive Programm.

DISKINSERTED und DISKREMOVED: Sollte der Benutzer eine Disk in irgendein Drive rein- oder rausschieben, so erhält Ihr Programm ein Signal zugesandt. Anmerkung: Wie bei NEWPREFS erhält jedes geeignete Programm ein Signal.

In unserem Programm wird das IDCMP-Flag CLOSEWINDOW gesetzt, da wir ganz gerne wissen wollen, wann der Benutzer das CloseWindow-Gadget gedrückt hat.

Flags: Hier werden verschiedene Feinheiten des Fensters beschrieben, z.B. welche System-Gadgets vorhanden sein sollen, wie das Fenster von Intuition verwaltet werden soll (bes. wann es aufgefrischt werden soll), ob es sofort aktiv sein soll, etc., etc. Dazu können folgende Flags gesetzt werden:

WINDOWSIZING: Versieht das Window mit dem Gadget, mit dem der Benutzer die Größe des Windows festlegen kann.

WINDOWDEPTH: Mit diesem Flag kann ein Fenster mit dem Gadget versehen werden, mit dem man bei übereinandergelegten Windows ein Window nach vorne bzw. hinten legen kann.

WINDOWCLOSE: Fügt das WindowClose-Gadget hinzu.

WINDOWDRAG: Bewirkt, daß die gesamte Title-Leiste des Windows in das Gadget verwandelt wird mit dem man das Window auf dem Screen hin und her bewegen kann.

GIMMEZEROZERO: Setzen Sie dieses Flag, wenn Sie ein Gimmezerozero-Window haben wollen. Bei diesem Windowtyp wird das Window in zwei Teile geteilt: In das eigentliche Window und in den gesamten Kram, wie Title-Leiste, Gadgets und Grenzen.

Dadurch wird gewährleistet, daß nur das eigentliche Window beschrieben werden kann, und der Rest von allen Grafik-Aktionen unberührt bleibt. Dieser Window-Typ verbraucht allerdings etwas mehr RAM.

SIMPLE_REFRESH: Ist dieses Flag gesetzt, so muß sich das Programm selbst darum kümmern, ob und wie es Teile erneuern will, wenn diese zum Beispiel durch andere Screens überdeckt wurden.

Ein Beispiel für diese Art des Windows ist das Programm Dots in der Schublade Demos auf der Workbench-Disk.

SMART_REFRESH: Hier kümmert sich Intuition automatisch um das Window. Wird es überlagert, so bringt Intuition den überlagerten Bereich in Sicherheit, und kopiert ihn nachher wieder zurück. Intuition scheitert nur, wenn der Benutzer die Größe des Windows verändert. Dann muß das Programm selbst einschreiten. Beispiele hierfür sind die Windows bei dem Programm Boxes in der Demoschublade, sowie die CLI-Windows.

SUPER_BITMAP: Dieses ist die speicheraufwendigste, jedoch auch die schönste Methode. Hierbei wird eine große Bitmap initialisiert und das Window stellt immer einen Teil davon da, je nach Größe des Windows. Ein Beispiel dafür ist das Programm Lines in der Schublade Demos. Verändern Sie die Größe des Windows einmal; jetzt sehen Sie, daß das ursprüngliche Window nur ein Teil eines riesigen Ganzen ist. Das Programm beschreibt immer die ganze Bitmap; Intuition stellt immer nur einen vom Benutzer ausgewählten Bereich dar.

BACKDROP: Macht das Window zu einem Fenster, das immer hinter allen anderen Fenstern liegt. Dabei ist egal, wann diese Fenster eröffnet werden, und ob sie vom Benutzer mittels der Gadgets nach hinten gelegt werden.

REPORTMOUSE: Setzen Sie dieses Flag, wenn Sie die Mousekoordinaten erhalten wollen.

BORDERLESS: Sorgt dafür, daß das Window ohne Ränder erscheint.

ACTIVATE: Sollte dieses Flag gesetzt sein, so wird Ihr Window sofort aktiv, sobald es geöffnet ist. Beachten Sie, wenn Sie dieses Flag setzen, daß dadurch unter Umständen die Eingaben des Benutzers umgeleitet werden können!

NOCAREREFRESH: Ist dieses Flag gesetzt, so sendet Intuition Ihnen keine Nachricht, wann das Window erneuert werden soll.

RMBTRAP: Setzen Sie dieses Flag, wenn Sie keine Menüoperationen haben wollen. Drückt der Benutzer jetzt die rechte Maustaste, so erhält Ihr Programm ein normales MOUSEBUTTON-Signal.

Wir wollten alle System-Gadgets, und die Methode SMART_REFRESH.

FirstGadget: Zeiger auf die Gadget-Struktur des ersten eigenen Gadgets, mit dem Sie das Window versehen wollen.

CheckMark: Zeiger auf die Image-Struktur des Zeichens, mit dem vom Benutzer ausgewählte Menüpunkte versehen werden sollen. Wollen Sie das Standard-Zeichen, so setzen Sie den Zeiger auf NULL.

Title: Titel des Winows. NULL, wenn Sie keinen Titel haben wollen.

Screen: Zeiger auf die Screen-Struktur des Screens, in dem dieses Window erscheinen soll. NULL, falls ein Standard-Screen (zum Beispiel Workbench-Screen) erwünscht ist.

BitMap: Zeiger auf eine BitMap-Struktur, falls SUPER_BITMAP gesetzt wurde.

MinWidth, MinHeight: Geben die Ausdehnung in Pixel an, die das Window mindestens haben sollte. Werden nur beachtet, wenn das WINDOSIZING-Flag gesetzt wurde.

MaxWidth, MaxHeight: Maximale Größe des Windows. Auch diese beiden Komponenten werden nur beachtet, wenn das WINDOWSIZING-Flag gesetzt wurde.

Type: Art des Screens, in dem das Window eröffnet werden soll. Momentan verfügbar sind folgende Flags:

WBENCHSCREEN: für Workbench-Screen
CUSTOMSCREEN: für eigenen Screens

Da wir einen eigenen Screen haben, wird in Zeile 60 diese Komponente auf CUSTOMSCREEN gesetzt.

So, damit wäre die NewWindow-Struktur geschafft. Nachdem alle Werte ordnungsgemäß eingespeichert worden sind, wird das Window mittels des Intuition-Befehls OpenWindow eröffnet. OpenWindow erwartet als Argument die Adresse einer NewWindow-Struktur.

Ist alles zur Zufriedenheit von Intuition abgelaufen, wird ein Zeiger auf eine Window-Struktur zurückgegeben. Ist irgendetwas Merkwürdiges geschehen, wird Null an den Aufrufer gegeben. Meistens wird das Programm dann eine entsprechende Meldung ausgeben und den Ablauf beenden. So auch unsere Demo.

Die Window-Struktur enthält folgendes:

NextWindow: Zeiger auf die Window-Struktur eines weiteren Windows innerhalb dieses Screens. Dadurch sind alle Windows miteinander verbunden.

LeftEdge, TopEdge, Width, Height: Ausmaße und Lage des Windows.

MouseY, MouseX: Koordinaten der Maus

MinWidth, MinHeight, MaxWidth, MaxHeight: Minimale bzw. maximale Ausdehnung des Fensters.

Flags: siehe NewWindow.Flags

MenuStrip: Zeiger auf die Menü-Struktur des ersten Menüs.

Title: Titel des Windows.

FirstRequester: Zeiger auf die Requester-Struktur des ersten Requesters für dieses Window.

DMRequest: Zeiger auf die Requester-Struktur der ersten Double-click-Requesters für dieses Window.

ReqCount: Zähler für die Anzahl der Requester, die dieses Window blockieren.

WScreen: Zeiger auf die Screen-Struktur dieses Screens.

RPort: Zeiger auf die RastPort-Struktur für dieses Window.

BorderLeft, BorderTop, BorderRight, BorderBottom: Beschreiben Aussehen des Randes.

BorderRPort: Zeiger auf RastPort-Struktur des Randes.

FirstGadget: Zeiger auf Gadget-Struktur des ersten Gadgets für dieses Window.

Parent, Descendant: Werden von Intuition beim Öffnen bzw. Schließen von Windows benutzt.

Pointer: Zeiger auf die Sprite-Datas für den Mauszeiger.

PtrHeight, PtrWidth: Höhe & Weite des Mauszeigers. Die Weite darf nicht größer als 16 sein.

XOffset, YOffset: Sprite-Offsets

IDCMPFlags: Hier kommen die Signale rein, die irgendwelche Aktionen des Benutzers anzeigen.

UserPort, WindowPort: Zeiger auf MsgPort-Strukturen. Diese Ports dienen zur Übertragung von Daten und werden in einem späteren Kapitel noch genauer beschrieben.

MessageKey: Zeiger auf IntuiMessage-Struktur. Dient zum Übertragen von Intuition-Messages.

DetailPen, BlockPen: wie üblich

CheckMark: Zeiger auf Image-Struktur, die das Zeichen beschreibt, mit dem ausgewählte Menüpunkte abgehakt werden. Ist dieser Zeiger auf NULL gesetzt, wird das Standardzeichen (der Haken) benutzt.

ScreenTitle: Namen, den der Screen erhält, wenn dieses Window aktiv ist.

GZZMouseX, GZZMouseY: Koordinaten der Maus bei Gimmezerozero-Windows.

GZZWidth, GZZHeight: Geben die Größe des wirklichen Windows bei Gzz-Windows an.

ExtData: ???

UserData: Zeiger auf irgendwelche Daten, die der Benutzer für wichtig hält.

WLayer: Zeiger auf Layer-Struktur für dieses Window. Enthält den gleichen Wert wie RPort->Layer innerhalb der Window-Struktur.

So, damit wären die benötigten Strukturen beendet. Weiter geht's in Zeile 71. Der Wait-Befehl setzt den Task in den Wartezustand. In diesem Fall wartet der Task einfach darauf, daß irgendein Signal am Userport angelegt wird. Da wir innerhalb der NewWindow-Struktur nur ein Signal angelegt haben (CLOSEWINDOW in Zeile 50), kann auch nur dieser Fall eingetreten sein: Der Benutzer hat das Closewindow-Gadget angeklickt.

Um den Speicherplatz wieder freizugeben, werden daraufhin das Window, der Screen und die Intuition-Library geschlossen. Schließlich wird das Programm beendet.

Alles klar? Gut, dann probieren Sie ein bißchen herum, indem Sie einige Werte verändern, und dann das ganze noch einmal compilieren, linken und laufen lassen. Schauen Sie sich dazu am besten noch einmal die obigen Erklärungen zu den vier Strukturen NewScreen, Screen, NewWindow und Window an.

Kommen wir zum nächsten Programm. Mit diesem wollen wir Ihnen zeigen, wie man einfach und dennoch effektiv einfache Grafikanwendungen programmieren kann. Zuerst einmal das Programm:

```
#include <exec/types.h>
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


main()
{
   struct Screen *Screen;
   struct Window *Window;
   struct NewScreen NewScreen;
   struct NewWindow NewWindow;
```

```
 IntuitionBase = (struct IntuitionBase *)
                          OpenLibrary ("intuition.library", 0)
;
 if (IntuitionBase == NULL)
 {
    printf ("Ups? Wo ist die Intuition-Library?\n");
    exit (FALSE);
 }


 GfxBase = (struct GfxBase *) OpenLibrary ("graphics.libra
ry", 0);
 if (GfxBase == NULL)
 {
    printf ("Hey, was macht die Grafik-Library?\n");
    exit (FALSE);
 }


 NewScreen.LeftEdge = 0; NewScreen.TopEdge = 0;
 NewScreen.Width = 320; NewScreen.Height = 200;
 NewScreen.Depth = 2;
 NewScreen.DetailPen = 0; NewScreen.BlockPen = 1;
 NewScreen.ViewModes = NULL;
 NewScreen.Type = CUSTOMSCREEN;
 NewScreen.Font = NULL;
 NewScreen.DefaultTitle = NULL;
 NewScreen.Gadgets = NULL;
 NewScreen.CustomBitMap = NULL;

 Screen = (struct Screen *) OpenScreen (&NewScreen);
 if (Screen == NULL)
 {
    printf ("Screen laesst sich nicht oeffnen!!!\n");
    exit (FALSE);
 }
```

```
NewWindow.LeftEdge = 0;
NewWindow.TopEdge = 0;
NewWindow.Width = 320;
NewWindow.Height = 200;
NewWindow.DetailPen = 0;
NewWindow.BlockPen = 1;
NewWindow.IDCMPFlags = NULL;
NewWindow.Flags = SMART_REFRESH | BACKDROP | BORDERLESS |
ACTIVATE |
                  NOCAREREFRESH;
NewWindow.FirstGadget = NULL;
NewWindow.CheckMark = NULL;
NewWindow.Title = NULL;
NewWindow.Screen = Screen;
NewWindow.BitMap = NULL;
NewWindow.MinWidth = NULL;
NewWindow.MinHeight = NULL;
NewWindow.MaxWidth = NULL;
NewWindow.MaxHeight = NULL;
NewWindow.Type = CUSTOMSCREEN;

Window = (struct Window *) OpenWindow (&NewWindow);
if (Window == NULL)
{
   printf ("Probleme mit dem Window...\n");
   exit (FALSE);
   }

   ShowTitle (Screen, FALSE);


   Move (Window->RPort, 160, 100);
   Window->MouseX = 160; Window->MouseY = 100;

   while ((Window->MouseX != 0) || (Window->MouseY != 0))
      Draw (Window->RPort, Window->MouseX, Window->MouseY);



   CloseWindow (Window);
   CloseScreen (Screen);
   CloseLibrary (GfxBase);
   CloseLibrary (IntuitionBase);
}
```

Nachdem Sie es zum Ausführen gebracht haben, können Sie mit der Maus auf dem Bildschirm malen. Hat keinen sehr großen Sinn, ist aber sehr gut zum Erklären. Das Programm ist dem vorhergehenden recht ähnlich. Es wird jedoch noch eine weitere Library eröffnet: die Grafik-Library. Dies geschieht in den Zeilen 26-31. Das Öffnen dieser Library geschieht ähnlich dem Öffnen der Intuition-Library, und braucht daher nicht weiter beschrieben zu werden.

Auch das Öffnen des Screens dürfte Ihnen jetzt klar sein. Wir haben diesmal allerdings die kleinere Auflösung von 320 * 200 Bildschirmpunkten gewählt.

Beim Window ist doch einiges anders geworden. Es wurden einige besondere Attribute gesetzt. Unser Window hat keine Ränder (BORDERLESS) und ist immer hinter allen anderen Windows (BACKDROP). Dadurch wirkt das Window fast wie ein Screen. Im Gegensatz zum Screen hat dieses Window jedoch einen entscheidenden Vorteil: Grafik-Befehle wirken sich immer noch auf die Bitmap des Windows aus, und nicht auf die Bitmap des Screens. Dadurch ist zum Beispiel gewährleistet, daß die Grafikbefehle keine anderen Windows überschreiben. Außerdem ist dadurch auch gewährleistet, daß keine anderen Windows die Grafiken einfach auslöschen können (das Window ist vom Typ SMART_REFRESH). Fazit: Grafikprogrammierung ist genauso effektiv wie die Low-Level-Grafikprogrammierung; jedoch viel einfacher, da man nicht auf so viele Fälle achten muß.

Der ShowTitle-Befehl in Zeile 81 sorgt dafür, daß die Titel-Zeile des Screens verschwindet. Und dann kommt ein Grafik-Befehl: Move. Dieser Befehl bewegt den Grafik-Zeiger auf die angegebenen Koordinaten. Als erstes Argument benötigt dieser Befehl einen Zeiger auf die RastPort-Struktur, in der der Grafik-Befehl seine Wirkung zeigen soll. Würden Sie hier den RastPort des Screens angeben, so würde der Grafik-Zeiger für den Screen-Rastport bewegt werden.

In der Zeile 88 wird geschaut, ob der Maus-Zeiger in die linke obere Ecke des Bildschirms bewegt wurde. Ist dies nicht der Fall, wird eine Linie von dem Punkt, wo sich der Grafik-Zeiger zuletzt befand, bis zu den aktuellen Maus-Koordinaten gezogen.

Bevor das Programm beendet wird, muß in den Zeilen 94-97 wieder alles geschlossen werden.

3.3.2 Routinen mit Intuition-Strukturen und -Funktionen

Hier finden Sie einige nützliche Routinen mit Intuition-Strukturen und Intuition-Funktionen. Die benötigten Strukturen werden direkt anhand der Listings erklärt.

3.3.2.1 FindScreen

Diese Routine besorgt Ihnen einen Zeiger auf die Screen-Struktur zu einem bestimmten Screen. Wollen Sie die Screen-Struktur des gerade aktiven Screens, so geben Sie der Routine als Argument eine 0. Wollen Sie die Screen-Struktur zu einem beliebigen Screen, so müssen Sie den Namen dieses Screens angeben.

Hier das Listing der Routine FindScreen. Zur Demonstration gleich mit einem Hauptprogramm:

```
#include <exec/types.h>
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;


struct Screen *FindScreen (name)
   char name[];
{
   struct Screen *screen;
```

```
   if (name == (char) 0)
      return (IntuitionBase->ActiveScreen);


   screen = IntuitionBase->ActiveScreen;

   while (screen->NextScreen != 0)
   {
      if (screen->Title == name)
         return (screen);

      screen = screen->NextScreen;
   }

   return (0);
}


main ()
{
   struct Screen *screen;


   IntuitionBase = (struct IntuitionBase *)

                              OpenLibrary ("intuition.library", 0)
  ;
   if (IntuitionBase == NULL)
   {
      printf ("Sorry, keine Intuition-Library fuer Dich.\n")
  ;
      exit (FALSE);
   }


   screen = FindScreen (0);


   if ((screen->Flags && WBENCHSCREEN) != NULL)
      printf ("Der aktive Screen ist der Workbench-Screen!!!
  \n");
   else printf ("Hm, irgendwas ist schief gelaufen...\n");
}
```

```
#ifndef INTUITION_INTUITIONBASE_H
#define INTUITION_INTUITIONBASE_H 1


/*** intuitionbase.h ****************************************
  *****************
 *
 *  the IntuitionBase structure and supporting structures
 *
 *          Modification History
 *      date    :   author :    Comments
 *     ------       ------      ------------------------------
  ----------
 *     3-1-85       -=RJ=-   created this file!
 *
 ***********************************************************
  *****************/
```

Mit unserem Lattice-Compiler V3.03 ließ sich dieses Programm nicht ordnungsgemäß compilieren. Der Compiler meldete, daß die Struktur IntuitionBase keine Komponente namens ActiveScreen besitzt. Ein Blick in das Headerfile intuition/intuitionbase.h ergab, daß die Struktur IntuitionBase tatsächlich nur die Komponente LibNode besaß. Nachdem wir das Headerfile jedoch so umgeändert hatten, wie es in dem ROM Kernel Manual Volume II zu finden ist, lief alles anstandslos. Falls also dieser Fehler bei Ihnen auch auftreten sollte, ändern Sie Ihr Headerfile so ab, daß es ungefähr dieses Aussehen hat:

```
#ifndef EXEC_LIBRARIES_H
#include "exec/libraries.h"
#endif

#ifndef GRAPHICS_VIEW_H
#include "graphics/view.h"
#endif

/*
 * Be sure to protect yourself against someone modifying the
  se data as
 * you look at them. This is done by calling:
 *
 * lock = LockIBase(0), which returns a ULONG. When done cal
  l
 * UnlockIBase (lock) where lock is what LockIBase() returne
  d.
 *
 * NOTE: these libraries functions are simply stubs now, but
   should be
 * called to be compatible with future releases.


 */

/* ==========================================================
  =============== */
/* === IntuitionBase ========================================
  =============== */
/* ==========================================================
  =============== */
struct IntuitionBase
{
    struct Library LibNode;

    struct View ViewLord;

    struct Window *ActiveWindow;
    struct Screen *ActiveScreen;
```

```
/* the FirstScreen variable points to the frontmost Screen.
   Screens are
 * then maintained in a front to back order using Screen.Nex
  tScreen
 */
    struct Screen *FirstScreen; /* for linked list of all sc
  reens */
};

#endif
```

So, und jetzt zu dem Programm. Das ganze Programm beruht auf der Struktur IntuitionBase. Diese Struktur enthält folgende Komponenten:

LibNode: Eine Struktur vom Typ Library.

ViewLord: Eine View-Struktur.

ActiveWindow: Ein Zeiger auf die Window-Struktur des gerade aktiven Windows.

ActiveScreen: Ein Zeiger auf die Screen-Struktur des gerade aktiven Screens.

FirstScreen: Ein Zeiger auf die Screen-Struktur des vordersten Screens.

Ist das Argument für FindScreen gleich 0, so gibt die Routine einfach den Inhalt von ActiveScreen an den Aufrufer zurück, da ja anscheinend die Screen-Structur des gerade aktiven Screens gefragt war. Wurde als Argument jedoch ein Name übergeben, geht FindScreen alle Screens, die ja durch die Komponente NextScreen innerhalb der Screen-Struktur miteinander verbunden sind, durch, bis es den Screen mit dem Namen findet.

Ist kein Screen mit diesem Namen zu finden, so gibt FindScreen eine 0 zurück, und der Aufrufer weiß, daß etwas nicht ganz geklappt hat. In main wird FindTask einmal kurz demonstriert.

3.3.2.2 FindWindow

Hier die gleiche Routine wie oben - diesmal jedoch für Windows statt für Screens.

```
#include <exec/types.h>
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;


struct Window *FindWindow (name)
   char name[];
{
   struct Window *window;


   if (name == (char) 0)
      return (IntuitionBase->ActiveWindow);


   window = IntuitionBase->ActiveWindow;

   while (window->NextWindow != 0)
   {
      if (window->Title == name)
         return (window);

      window = window->NextWindow;
   }

   return (0);
}


main ()
{
   struct Window *window;


   IntuitionBase = (struct IntuitionBase *)
```

```
                         OpenLibrary ("intuition.library", 0)
;
  if (IntuitionBase == NULL)
  {
     printf ("Sorry, keine Intuition-Library fuer Dich.\n")
;
     exit (FALSE);
  }


  window = FindWindow (0);

  printf ("Name des aktiven Screens: %s\n", window->Title);



  window = FindWindow ("Gonzo");
  if (window == 0)
     printf ("Gonzo ist verschollen...\n");
  else printf ("Bitte? Gonzo ist bei Ihnen? Sofort melden!\
n");


  CloseLibrary (IntuitionBase);
}
```

Da das Programm dem obrigen absolut ähnlich ist, braucht es wohl nicht näher erläutert zu werden.

3.3.2.3 MoveScreen, MoveWindow u.a. Intuition-Routinen

Die Intuition-Library enthält zahlreiche sehr interessante Routinen. Eine davon ist MoveScreen. Wie Sie vielleicht schon am

Namen erkannt haben, läßt sich damit ein Screen verschieben. Die Funktion hat folgenden Syntax:

```
MoveScreen (screen, dx, dy)
```

Das Argument Screen ist ein Zeiger auf die Screen-Struktur des Screens, der verschoben werden soll. Die Anzahl der Pixel, um die verschoben werden soll, ist in dx und dy zu finden. Zur Zeit ist jedoch eine Verschiebung des Screens in X-Richtung nicht möglich; folglich wird auch die dx-Variable nicht berücksichtigt. Geben Sie am besten immer 0 dafür an, damit Ihr Programm auch auf weiteren Versionen noch läuft.

Hier ein kleines Programm, das eine Anwendung des Move-Screen-Befehls zeigt:

```
#include <exec/types.h>
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;



void PrintText (RPort, string, x, y)
   struct RastPort *RPort;
   char string[];
   LONG x, y;
{
   Move (RPort, x, y);
   Text (RPort, string, strlen (string));
}
```

```
void Text_Ausgabe (RPort)
   struct RastPort *RPort;
{
   PrintText (RPort, "Hier sehen Sie ein kleines Demo.", 150
  , 10);
   PrintText (RPort, "Es wird demonstriert, wie man mit weni
  gen Befehlen", 60, 40);
   PrintText (RPort, "einen Scroll-Effekt erzielen kann.", 6
  0, 60);
   PrintText (RPort, "Setzen Sie dieses in Ihren eigenen Pro
  grammen ein,", 60, 80);
   PrintText (RPort, "und Sie werden dieses Programm merklic
  h aufbessern.", 60, 100);
   PrintText (RPort, "Es gruesst recht herzlich", 100, 130);

   PrintText (RPort, "Horny", 150, 150);
   PrintText (RPort, "P.S.: Gruesse auch an Lotta, Gonzo, Do
  pe & Team 3!!!", 30, 180);
}

main()
{
   struct Screen *Screen;
   struct Window *Window;
   struct NewScreen NewScreen;
   struct NewWindow NewWindow;
   UCOUNT zaehler;


   IntuitionBase = (struct IntuitionBase *)
                      OpenLibrary ("intuition.library", 0)
  ;
   if (IntuitionBase == NULL)
   {
      printf ("Ich kann leider die Intuition-Library nicht f
  inden.\n");
      exit (FALSE);
   }
```

```
 GfxBase = (struct GfxBase *) OpenLibrary ("graphics.libra
ry", 0);
 if (GfxBase == NULL)
 {
    printf ("Ich kann leider die Grafik-Bibliothek nicht f
inden.\n");
    exit (FALSE);
 }


 NewScreen.LeftEdge = 0; NewScreen.TopEdge = 0;
 NewScreen.Width = 640; NewScreen.Height = 200;
 NewScreen.Depth = 2;
 NewScreen.DetailPen = 0; NewScreen.BlockPen = 1;
 NewScreen.ViewModes = HIRES;
 NewScreen.Type = CUSTOMSCREEN;
 NewScreen.Font = NULL;
 NewScreen.DefaultTitle = NULL;
 NewScreen.Gadgets = NULL;
 NewScreen.CustomBitMap = NULL;

 Screen = (struct Screen *) OpenScreen (&NewScreen);
 if (Screen == NULL)
 {
    printf ("Ich kann leider den Screen nicht oeffnen.\n")
;
    exit (FALSE);
 }
```

```
 NewWindow.LeftEdge = 0;
 NewWindow.TopEdge = 0;
 NewWindow.Width = 640;
 NewWindow.Height = 200;
 NewWindow.DetailPen = 0;
 NewWindow.BlockPen = 1;
 NewWindow.IDCMPFlags = NULL;
 NewWindow.Flags = SMART_REFRESH | BACKDROP | BORDERLESS |
 ACTIVATE |
                  NOCAREREFRESH;
 NewWindow.FirstGadget = NULL;
 NewWindow.CheckMark = NULL;
 NewWindow.Title = NULL;
 NewWindow.Screen = Screen;
 NewWindow.BitMap = NULL;
 NewWindow.MinWidth = NULL;
 NewWindow.MinHeight = NULL;
 NewWindow.MaxWidth = NULL;
 NewWindow.MaxHeight = NULL;
 NewWindow.Type = CUSTOMSCREEN;

 Window = (struct Window *) OpenWindow (&NewWindow);
 if (Window == NULL)
 {
    printf ("Ich kann leider das Window nicht oeffnen.\n")
;
    exit (FALSE);
 }
   ShowTitle (Screen, FALSE);



   MoveScreen (Screen, 0, 200);


   Text_Ausgabe (Window->RPort);


   for (zaehler = 0; zaehler < 200; zaehler++)
      MoveScreen (Screen, 0, -1);


   Delay (200);



   CloseWindow (Window);
   CloseScreen (Screen);
   CloseLibrary (GfxBase);
   CloseLibrary (IntuitionBase);
}
```

Innerhalb von main verläuft zuerst einmal alles wie gewohnt. Es wird die Intuition- sowie die Grafik-Library eröffnet. Danach wird ein Screen und ein Window innerhalb dieses Screens eröffnet. In Zeile 112 tritt die sagenhafte Funktion MoveScreen schließlich auf. Mit MoveScreen (0, 200) wird der Screen um 200 Pixel nach unten verschoben. Da der Screen nur 200 Pixel hoch ist, verschwindet er damit vom Bildschirm.

In Zeile 115 wird die Funktion Text_Ausgabe aufgerufen. Diese ist in den Zeilen 20 - 31 zu finden. Diese Routine benutzt eine andere Routine: PrintText. PrintText ist in den Zeilen 10 - 17 zu finden. Mit PrintText kann ein Text auf einem bestimmten Rastport ausgegeben werden. PrintText benötigt vier Argumente:

RPort: Zeiger auf die RastPort-Struktur, in der der Text erscheinen soll.

String: Auszugebener String

X, Y: Koordinaten, an denen der String ausgegeben werden soll.

PrintText benutzt zwei Grafik-Funktionen: Move und Text. Mit Move wird der Grafik-Pointer an eine bestimmte Stelle innerhalb eines bestimmten Rastports gesetzt. Die Funktion Text gibt schließlich den gewünschten Text aus. Text verlangt drei Argumente.

1. Einen Zeiger auf die Rastport-Struktur.
2. Den auszugebenen Text.
3. Die Länge des auszugebenden Textes.

Die Länge des auszugebenden Textes wird in unserem Programm mittels der C-Funktion strlen bestimmt.

Nachdem der Text in das Window geschrieben wurde, wird der Screen wieder nach oben geholt. Dies geschieht in den Zeilen 118/119. Hier wird 200 mal der Befehl "MoveScreen (Screen, 0, -1)" ausgeführt. Dadurch wird der Screen 200 mal um einen Pi-

xel nach oben verschoben. Danach ist er wieder auf seinem alten Platz; das Programm wartet noch einmal ca. 4 Sekunden, schließt dann alles wieder ordnungsgemäß und verabschiedet sich.

Es gibt auch eine entsprechende Funktion für Windows: MoveWindow. Der Syntax und die Wirkungsweise ist ähnlich wie bei MoveScreen - nur wird hier ein Window verschoben, und so muß auch ein Zeiger auf eine Window-Struktur übergeben werden. In dem Zusammenhang seien noch acht weitere Intuition-Routinen erwähnt:

SetWindowTitles:

Syntax:	SetWindowTitles (Window, WindowTitle, ScreenTitle)
Window:	Zeiger auf Window-Struktur
WindowTitle:	Zeiger auf String, der der neue Titel des Windows sein soll. Bei 0 kein Titel, und bei -1 bleibt der alte Titel.
ScreenTitle:	wie oben, jedoch für Screentitel

SizeWindow:

Syntax:	SizeWindow (Window, dx, dy)
Window:	Zeiger auf Window-Struktur
dx, dy:	Anzahl der Pixel, um die das Window in X- bzw. Y-Richtung vergrößert oder verkleinert werden soll.

OpenWorkBench:

Syntax:	success = OpenWorkBench ()
success:	TRUE, wenn alles glatt ging; FALSE, wenn ein Fehler auftauchte. Eröffnet den Workbench-Screen, falls noch nicht offen.

CloseWorkBench:

Syntax: success = CloseWorkBench ()
success: siehe oben, schließt den Workbench-Screen, falls noch offen.

WBenchToBack:

Syntax: success = WBenchToBack ()
success: siehe oben, bringt den Workbench-Screen hinter alle anderen Screens.

WBenchToFront:

Syntax: success = WBenchToFront ()
success: siehe oben, bringt den Workbench-Screen vor alle anderen Screens.

WindowToBack:

Syntax: WindowToBack (Window)
Window: Zeiger auf die Window-Struktur des Windows, das hinter alle anderen Windows auf dem aktuellen Screen gebracht werden soll.

WindowToFront:

Syntax: WindowToFront (Window)
Window: Zeiger auf die Window-Struktur für das Window, das vor alle anderen Windows gebracht werden soll.

3.3.2.4 DisplayBeep oder wie lasse ich Screens blinken

Bei den meisten Computern gibt es einen CHR$-Wert, der einen sehr interessanten Effekt bewirkt. Das ist CHR$(7) und er gibt einen Ton über die Lautsprecher aus.

Der Amiga besitzt eine Intuition-Funktion, die eine ähnliche Wirkung zeigt. Das ist die Funktion DisplayBeep. DisplayBeep benötigt ein Argument. Dieses Argument muß ein Zeiger auf die Screen-Struktur des Screens sein, der blinken soll. Ist dieses Argument 0, so läßt Intuition alle Screens einmal aufblinken.

Schauen Sie sich einmal diese kurze Demo an:

```
#include <exec/types.h>
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;


main ()
{
   int wert;


   IntuitionBase = (struct IntuitionBase *)
                            OpenLibrary ("intuition.library", 0)
  ;
   if (IntuitionBase == NULL)
   {
      printf ("Keine Intuition-Library, also kein Programm.\
  n");
      exit (FALSE);
   }
```

```
   printf ("Geben Sie einen Wert von 0 - 10 ein: ");
   scanf ("%d", &wert);

   if ((wert < 0) || (wert > 10))
   {
      DisplayBeep (IntuitionBase->ActiveScreen);
      printf ("Wert nicht in den angegebenen Grenzen!!!\n");

   }
   else
      printf ("Das Quadrat von %d ist ungefaehr %d.\n", wert
  , wert*wert);


   CloseLibrary (IntuitionBase);
}
```

Ist der eingegebene Wert nicht in den angegebenen Grenzen, so macht Intuition den Benutzer mittels DisplayBeep sofort darauf aufmerksam. DisplayBeep sollte immer dort eingesetzt werden, wo die es nicht reicht, die Aufmerksamkeit des Benutzers durch normale Textausgabe zu erregen, und wo ein Alert zu viel wäre.

Weitere denkbare Anwendungen für DisplayBeep wären zum Beispiel, den Screen am Ende eines zeitintensiven Programmes blinken zu lassen oder wenn bei einem Terminal-Programm eine Nachricht eingegangen ist.

3.3.3 Keine graue Maus mehr

Haben Sie immer noch den normalen Mauspointer? Den roten Pfeil? Oder haben Sie schon mit Preferences einen eigenen Mauszeiger konstruiert? Einen Luftballon? Ein Fadenkreuz? Wahrscheinlich.

Doch nicht nur der Benutzer ändert den Mauszeiger. Oft ist es auch recht sinnvoll und nett anzusehen, wenn ein Programm einen individuellen Zeiger besitzt. Denken Sie zum Beispiel an Graphicraft & Textcraft.

Ändern kann man den Mauszeiger recht einfach. Die Intuition-Library verfügt nämlich über einen Befehl, der genau dieses vollführt: SetPointer.

Zeigen wir die Funktion wieder anhand eines Beispiels:

```
#include <exec/types.h>
#include <exec/memory.h>
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;

UWORD spritedata[] =
{
   0, 0,
   0xffff, 0xffff,
   0xffff, 0xffff,
   0xffff, 0xffff,
   0x0000, 0xffff,
   0x0000, 0xffff,
   0x0000, 0xffff,
   0xffff, 0x0000,
   0xffff, 0x0000,
   0xffff, 0x0000,
   0, 0
};
```

```
main ()
{
   struct Screen *Screen;
   struct Window *Window;


   IntuitionBase = (struct IntuitionBase *)
                        OpenLibrary ("intuition.library", 0)
  ;
   if (IntuitionBase == NULL)
   {
      printf ("Was soll ich machen? Keine Intuition-Library!
  \n");
      exit (FALSE);


   }

   GfxBase = (struct GfxBase *) OpenLibrary ("graphics.libra
  ry", 0);
   if (GfxBase == NULL)
   {
      printf ("Und nun? Ich kann die Graphics-Library nicht
  finden!\n");
      CloseLibrary (IntuitionBase);
      exit (FALSE);
   }


   Screen = IntuitionBase->ActiveScreen;
   Window = IntuitionBase->ActiveWindow;


   SetPointer (Window, spritedata, 9, 16, -7, -5);

   SetRGB4 (Screen->ViewPort, 17, 10, 10,  0);
   SetRGB4 (Screen->ViewPort, 18, 15,  0,  0);
   SetRGB4 (Screen->ViewPort, 19,  0,  0,  0);


   CloseLibrary (IntuitionBase);
   CloseLibrary (GfxBase);


   Wait (0);
}
```

Wunderschön, nicht wahr?

Zur Erklärung:
Die Intuition-Funktion SetPointer gibt kein Resultat zurück, erwartet jedoch 5 Argumente:

Window:	Zeiger auf die Window-Struktur des Windows, dessen Mauszeiger geändert werden soll.
Pointer:	Zeiger auf die Daten für den Sprite.
Height:	Höhe des Mauszeigers in Zeilen.
Width:	Breite des Mauszeigers in Pixels (nicht größer als 16).
XOffset:	Gibt die X-Koordinate des "hot spots" an. Der "hot spot" ist der Punkt, der den eigentlichen Mauszeiger darstellt, d. h., die Stelle, mit der Sie alles anklicken müssen.
YOffset:	Y-Koordinate des hot-spots.

Der Aufruf von SetPointer geschieht in dem Demo-Programm in Zeile 56. Das Argument Window wurde in Zeile 52 mit einem Zeiger auf die Window-Struktur des gerade aktuellen Windows belegt.

Mit spritedata wird an SetPointer die Adresse des Feldes für die Daten des neuen Zeigers übergeben. Dieses Feld muß folgendermaßen aufgebaut sein:

1. Zwei Nullen am Anfang.

2. Die eigentlichen Spritedaten. Hierbei braucht jede Sprite-Zeile zwei Worte (16-Bit-Werte), da insgesamt vier Farben zur Verfügung stehen (besser: drei Farben plus durchscheinend). Legt man die beiden Worte übereinander, so ergibt sich die eigentliche Farbe.

   1. Beispiel:
   Wort 1: $ffff (%1111111111111111)

Wort 2: $ffff (%1111111111111111)
Bei %11 wird die Farbe aus dem Farbregister 19 genommen.

2. Beispiel:
Wort 1: $0000 (%0000000000000000)
Wort 2: $ffff (%1111111111111111)
Bei %10 wird Farbregister 18 genommen.

Entsprechend wird bei der Kombination %01 das Farbregister 17, und bei der Kombination %00 die Hintergrundfarbe genommen.

3. Wieder zwei Nullen zum Schluß. Die Argumente 9 und 16 beim Aufruf von SetPointer geben die Größe des neuen Mauszeigers an. In diesem Fall ist er 9 Pixel lang und 16 Pixel breit.

Mit -7 und -5 wird schließlich der eigentliche Mauszeiger, der sog. "hot spot", ziemlich genau in die Mitte des neuen Mauszeiger(-Bildes) gelegt.

In den Zeilen 58 - 60 werden schließlich die Farbregister 17 - 19, die wie oben aufgeführt für die Farbe des Mauszeigers verantwortlich sind, mit den gewünschten Farben geladen. Dazu bedienen wir uns des Grafik-Befehls SetRGB4. Dieser Befehl benötigt erst einmal einen Zeiger auf den Viewport, für den die Farben bestimmt werden sollen. Diesen Viewport erhalten wir aus der Screen-Struktur des gerade aktuellen Screens. Weiterhin benötigt SetRGB4 die Nummer des gewünschten Farbregisters.

Und schließlich sind noch die gewünschten Rot-, Grün- und Blau-Anteile gefordert. Diese bestehen aus 4 Bits (können also Werte von 0 - 15 durchlaufen), und geben die Intensität des jeweiligen Anteils an. Wenn alle auf 0 sind, ist schwarz angesagt; sind alle auf 15, so wird weiß dargestellt.

Es gibt auch ein Gegenstück zu Setpointer: ClearPointer. Mittels dieser Intuition-Funktion kann eine Defintion eines Mauszeigers wieder rückgängig gemacht werden. Intuition stellt den Mauszeiger automatisch wieder auf den Standard-Mauszeiger zurück.

Das ganze im Programm:

```
#include <exec/types.h>
#include <intuition/intuition.h>


#define STANDARDPOINTER 1
#define CUSTOMPOINTER   0



struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;

UWORD spritedata[] =
{
   0, 0,
   0xffff, 0x0e00,
   0xffff, 0x0e00,
   0xfffc, 0x0e00,
   0xfffc, 0xfffc,
   0xfff0, 0xfff0,
   0xfffc, 0xfffc,
   0xfffc, 0x0e00,
   0xffff, 0x0e00,
   0xffff, 0x0e00,
   0, 0
};
```

```
main ()
{
   struct Screen *Screen;
   struct Window *Window;
   BOOL flag = STANDARDPOINTER;


   IntuitionBase = (struct IntuitionBase *)
                         OpenLibrary ("intuition.library", 0)
  ;
   if (IntuitionBase == NULL)


   {
      printf ("???\n");
      exit (FALSE);
   }

   GfxBase = (struct GfxBase *) OpenLibrary ("graphics.libra
  ry", 0);
   if (GfxBase == NULL)
   {
      printf ("??????\n");
      CloseLibrary (IntuitionBase);
      exit (FALSE);
   }



   Screen = IntuitionBase->ActiveScreen;
   Window = IntuitionBase->ActiveWindow;


   FOREVER
   {
      if (flag == STANDARDPOINTER)
      {
         flag = CUSTOMPOINTER;
         SetPointer (Window, spritedata, 9, 16, -7, -5);
      }
      else
      {
         flag = STANDARDPOINTER;
         ClearPointer (Window);
      }

      Delay (25);
   }
}
```

```
#include <stdio.h>
#include <exec/types.h>
#include <intuition/intuition.h>


struct GfxBase *GfxBase;
struct IntuitionBase *IntuitionBase;


main()
{
   struct View *view;
   struct ViewPort *viewport;
   COUNT zaehler;
```

Alles vor der Zeile 59 ist nur Initialisierung. In Zeile 59 startet schließlich das Hauptprogramm. In diesem wird mittels eines Flags immer zwischen zwei Anweisungen hin- und hergesprungen. Dadurch wird abwechselnd der neue Mauszeiger und der Standard-Mauszeiger gesetzt. Zwischendurch wird immer mit der AmigaDOS-Funktion Delay eine halbe Sekunde gewartet.

Weiter oben wurde gesagt, daß die Farbe für den Pointer in den Farb-Registern 17 - 19 zu finden ist. Kreativ, wie wir sind, fällt uns natürlich auch dazu gleich ein kleines Programm ein:

```
  GfxBase = (struct GfxBase *) OpenLibrary ("graphics.libra
 ry", 0);
  if (GfxBase == NULL)
  {
     printf ("Diesmal finde ich keine Grafik-Bibliothek.\n"
 );
     exit (FALSE);
  }

  IntuitionBase = (struct IntuitionBase *)
                      OpenLibrary ("intuition.library", 0)
 ;
  if (IntuitionBase == NULL)
  {
     printf ("Diesmal finde ich keine Intuition-Bibliothek.
 \n");
     exit (FALSE);
  }


  view = (struct View *) ViewAddress ();
  viewport = view->ViewPort;

  FOREVER
  {
     for (zaehler = 0; zaehler < 16; zaehler++)
     {
        SetRGB4 (viewport, 17, zaehler, zaehler, zaehler);
        Delay (20);
     }
  }
}
```

Noch ein Tip zum Schluß: Gestalten Sie Ihren Mauszeiger doch einmal so richtig interessant, indem Sie ein wenig Animation mit ins Spiel bringen. Dazu brauchen Sie nur einige verschiedene Sprite-Daten zu initialisieren, und diese dann nacheinander anstelle des Pointers zu setzen. Wenn diese Daten gut gewählt wurden, kann man schon sehr schöne Effekte erzielen; z.B. ein laufendes Männchen, oder dergleichen.

3.3.4 Text ausblenden - ganz einfach

Haben Sie schon den Amiga-Tutor laufen lassen? Ist Ihnen dabei aufgefallen, daß oft bestimmte Teile, sei es Grafik oder Text,

langsam aus- bzw. eingeblendet werden? Sieht ziemlich gut aus, und ist auf dem Amiga auch ziemlich gut zu programmieren.

Schauen Sie sich zu diesem Thema einmal folgendes Programm an:

```
#include <exec/types.h>
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;



main ()
{
   struct Screen *Screen;
   struct Window *Window;
   struct NewScreen NewScreen;
   struct NewWindow NewWindow;
   COUNT zaehler_1, zaehler_2;



   IntuitionBase = (struct IntuitionBase *)
                           OpenLibrary ("intuition.library", 0)
  ;
   if (IntuitionBase == NULL)
   {
      printf ("Grummel, grummel, grummel...\n");
      exit (FALSE);
   }


   GfxBase = (struct GfxBase *) OpenLibrary ("graphics.libra
  ry", 0);
   if (GfxBase == NULL)
   {
      printf ("Grummel, grummel, grummel, grummel, grummel..
   \n");
      CloseLibrary (IntuitionBase);
      exit (FALSE);
   }
```

```
NewScreen.LeftEdge = 0; NewScreen.TopEdge = 0;
NewScreen.Width = 320; NewScreen.Height = 200;
NewScreen.Depth = 2;
NewScreen.DetailPen = 0; NewScreen.BlockPen = 1;
NewScreen.ViewModes = NULL;
NewScreen.Type = CUSTOMSCREEN;
NewScreen.Font = NULL;
NewScreen.DefaultTitle = NULL;
NewScreen.Gadgets = NULL;
NewScreen.CustomBitMap = NULL;

Screen = (struct Screen *) OpenScreen (&NewScreen);
if (Screen == NULL)
{
   printf ("DU bekommst leider keinen Screen!\n");
   CloseLibrary (IntuitionBase);
   CloseLibrary (GfxBase);
   exit (FALSE);
}


NewWindow.LeftEdge = 0; NewWindow.TopEdge = 0;
NewWindow.Width = 320; NewWindow.Height = 200;
NewWindow.DetailPen = 0; NewWindow.BlockPen = 1;
NewWindow.IDCMPFlags = NULL;
NewWindow.Flags = SMART_REFRESH ¦ BACKDROP ¦ BORDERLESS ¦
ACTIVATE ¦
                  NOCAREREFRESH;
NewWindow.FirstGadget = NULL;
NewWindow.CheckMark = NULL;
NewWindow.Title = NULL;
NewWindow.Screen = Screen;
NewWindow.BitMap = NULL;
NewWindow.MinWidth = NULL; NewWindow.MinHeight = NULL;
NewWindow.MaxWidth = NULL; NewWindow.MaxHeight = NULL;
NewWindow.Type = CUSTOMSCREEN;

Window = (struct Window *) OpenWindow (&NewWindow);
```

```
if (Window == NULL)
{
   printf ("Keinen Bock auf `nen neues Window.\n");
   CloseScreen (Screen);
   CloseLibrary (IntuitionBase);
   CloseLibrary (GfxBase);
   exit (FALSE);
}


ShowTitle (Screen, FALSE);



SetDrMd (Window->RPort, JAM1);
SetAPen (Window->RPort, 1);
SetRGB4 (&(Screen->ViewPort), 0, 0, 0, 0);
SetRGB4 (&(Screen->ViewPort), 1, 0, 0, 0);

Move (Window->RPort, 10, 60);
Text (Window->RPort, "It's so dreamy _ oh fantasy free me
", 35);
Move (Window->RPort, 20, 80);
Text (Window->RPort, "So you can't see me _ oh, not at al
l", 36);
Move (Window->RPort, 30, 120);
Text (Window->RPort, "The Rocky Horror Picture Show", 29)
;

for (zaehler_1 = 1; zaehler_1 < 4; zaehler_1++)
{
   Delay (20);

   for (zaehler_2 = 0; zaehler_2 < 16; zaehler_2++)
   {
      SetRGB4 (&(Screen->ViewPort), 1, zaehler_2, zaehler
_2, zaehler_2);
      Delay (4);
   }
```

```
      for (zaehler_2 = 15; zaehler_2 >= 0; zaehler_2--)
      {
         SetRGB4 (&(Screen->ViewPort), 1, zaehler_2, zaehler
   _2, zaehler_2);
         Delay (4);
      }
   }


   CloseWindow (Window);
   CloseScreen (Screen);
   CloseLibrary (GfxBase);
   CloseLibrary (IntuitionBase);
}
```

Gesehen? Gut, dann ein bißchen Theorie:

Jeder Bildschirm besteht beim Amiga aus 1-6 Ebenen, sog. Bitplanes. Die Anzahl der Ebenen bestimmt die Anzahl der verfügbaren Farben. Ist zum Beispiel nur eine Ebene vorhanden, so kann ein Bildschirmpixel entweder 0 oder 1 sein; es können also nur zwei Farben dargestellt werden.

Sind dagegen vier Ebenen vorhanden, so gibt es 2^4 = 16 Möglichkeiten für ein Bildschirmpixel, und es können demnach 16 Farben dargestellt werden.

Nach dieser Erklärung könnte man meinen, daß die Farbe eines Pixels auf dem Bildschirm unwiderruflich festgelegt it. Falsch! Die Bits in den verschiedenen Bitplanes geben nur an, welches Farbregister für dieses Pixel genommen werden soll. Stehen z. B. zwei Bitplanes zur Verfügung, und in beiden Bitplanes ist das linke obere Bit gleich 1, so wird dieses Pixel in der momentanen Farbe des Farbregisters %11 = 3 dargestellt.

Verändert man nun den Inhalt dieses Farbregisters, so ändern alle Bildschirmpixel, die auf dieses Farbregister zurückgreifen, automatisch ihre Farbe.

Will man nun einen Text oder eine Grafik aus- oder einblenden, so muß man nur den Inhalt des Farbregisters, das für den Text oder die Grafik zuständig ist, langsam dem Inhalt des Farbregisters, das für den Hintergrund zuständig ist, angleichen.

Und genau das vollführt das Beispielprogramm:

Zuerst wird der ganze benötigte Kram geöffnet. Dazu gehört die Intuition-Library, die Grafik-Library, ein geeigneter Screen, und ein Window. Nachdem dies alles ordnungsgemäß beendet wurde, wird in Zeile 91 der Zeichenmodus gesetzt. Sodenn wird in der darauffolgenden Zeile festgelegt, daß die Zeichenfarbe aus dem Farbregister 1 zu holen ist. Die nächsten zwei Zeilen sorgen dafür, daß sowohl das Farbregister 0 (Hintergrundfarbe) als auch das Farbregister 1 (Zeichenfarbe) mit der wunderhübschen Farbe schwarz belegt werden.

Die Zeilen 96 - 101 geben einen Text auf das Window aus - da die Zeichenfarbe jedoch gleich der Hintergrundfarbe ist, ist für den unbeteiligten Zuschauer noch nichts zu erspähen.

Die äußere Schleife ab Zeile 103 sorgt dafür, daß der gesamte Vorgang drei mal wiederholt wird. Die beiden inneren Schleifen sorgen dafür, daß die Farbregler langsam in Richtung weiß, und

danach wieder langsam in Richtung schwarz geschoben werden. Dadurch taucht der Text aus dem Schwarz hervor, geht alle möglichen Graustufen durch, bis zum strahlensten Weiß, um hiernach wieder ins Grau und schließlich ins Schwarz zu versinken.

Mit Delay wird verhindert, daß das ganze zu schnell abläuft.

3.3.5 Ein einfaches Malprogramm mit Menüs, Requesters und Rubberband

Hier haben wir etwas ganz besonderes für Sie: ein Malprogramm. Natürlich, es ist kein DeluxePaint. Doch dafür verfügen Sie über das Listing, und können es so ganz einfach Ihren Bedürfnissen anpassen, und 'ne ganze Menge für Ihre eigenen Programme lernen.

```
#include <exec/types.h>
#include <exec/exec.h>
#include <graphics/copper.h>
#include <graphics/gels.h>
#include <graphics/gfxbase.h>
#include <graphics/regions.h>
#include <intuition/intuition.h>
#include <devices/keymap.h>
#include <hardware/blit.h>


#define UP 0
#define DOWN 1
```

```
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;



void drawbox (Window, start_x, start_y, end_x, end_y)
   struct Window *Window;
   SHORT start_x, start_y, end_x, end_y;
{
   Move (Window->RPort, end_x, end_y);
   Draw (Window->RPort, end_x, start_y);
   Draw (Window->RPort, start_x, start_y);
   Draw (Window->RPort, start_x, end_y);
   Draw (Window->RPort, end_x, end_y);
}



main ()
{
   struct Screen *Screen;
   struct Window *Window;
   struct NewScreen NewScreen;
   struct NewWindow NewWindow;
   struct MenuItem MenuItem[8];
   struct IntuiText MenuText[4], BodyText, PositiveText, Neg
  ativeText;
   struct Image MenuImage[4];
   struct Menu Menu[2];
   struct IntuiMessage *Message;

   COUNT z;
   ULONG Class;
   USHORT Code;
   SHORT start_x, start_y, end_x, end_y;
   BOOL endflag = TRUE;
   BOOL penflag = UP;
   BOOL mousemoveflag = FALSE;
   BOOL drawflag = TRUE;
   BOOL boxflag = FALSE;
   BOOL rubberbandflag = FALSE;
```

```
 IntuitionBase = (struct IntuitionBase *)
                      OpenLibrary ("intuition.library", 0)
;
 if (IntuitionBase == NULL)
 {
    printf ("Tut mir leid. Keine Intuition-Library.\n");
    exit (FALSE);
 }


 GfxBase = (struct GfxBase *) OpenLibrary ("graphics.libra
ry", 0);
 if (GfxBase == NULL)
 {
    printf ("Tut mir ausserordentlich leid. Keine Grafik-L
ibrary.\n");
    CloseLibrary (IntuitionBase);
    exit (FALSE);
 }

NewScreen.LeftEdge = 0; NewScreen.TopEdge = 0;
NewScreen.Width = 320; NewScreen.Height = 200;
NewScreen.Depth = 2;
NewScreen.DetailPen = 0; NewScreen.BlockPen = 1;
NewScreen.ViewModes = NULL;
NewScreen.Type = CUSTOMSCREEN;
NewScreen.Font = NULL;
NewScreen.DefaultTitle = NULL;
NewScreen.Gadgets = NULL;
NewScreen.CustomBitMap = NULL;

Screen = (struct Screen *) OpenScreen (&NewScreen);
if (Screen == NULL)
{
   printf ("Ich bin untroestlich. Kein Screen.\n");
   CloseLibrary (GfxBase);
   CloseLibrary (IntuitionBase);
   exit (FALSE);
}
```

```
 NewWindow.LeftEdge = 0; NewWindow.TopEdge = 0;
 NewWindow.Width = 320; NewWindow.Height = 200;
 NewWindow.DetailPen = 0; NewWindow.BlockPen = 1;
 NewWindow.IDCMPFlags = MOUSEBUTTONS | MOUSEMOVE | MENUPIC
K;
 NewWindow.Flags = SMART_REFRESH | BACKDROP | BORDERLESS |
 ACTIVATE |
                  NOCAREREFRESH | REPORTMOUSE;
 NewWindow.FirstGadget = NULL;
 NewWindow.CheckMark = NULL;
 NewWindow.Title = NULL;
 NewWindow.Screen = Screen;
 NewWindow.BitMap = NULL;
 NewWindow.MinWidth = NULL; NewWindow.MinHeight = NULL;
 NewWindow.MaxWidth = NULL; NewWindow.MaxHeight = NULL;
 NewWindow.Type = CUSTOMSCREEN;

Window = (struct Window *) OpenWindow (&NewWindow);
if (Window == NULL)
{
   printf ("Ich bin schockiert. Kein Window.\n");
   CloseScreen (Screen);
   CloseLibrary (GfxBase);
   CloseLibrary (IntuitionBase);
   exit (FALSE);
}


ShowTitle (Screen, FALSE);
```

```
for (z = 0; z < 4; z++)
{
   MenuItem[z].LeftEdge = 0; MenuItem[z].TopEdge = z * 10
;
   MenuItem[z].Width = 64; MenuItem[z].Height = 10;
   MenuItem[z].Flags = ITEMTEXT ¦ ITEMENABLED ¦ HIGHBOX;
   MenuItem[z].MutualExclude = NULL;
   MenuItem[z].ItemFill = (APTR) &MenuText[z];
   MenuItem[z].SelectFill = NULL;
   MenuItem[z].Command = NULL;
   MenuItem[z].SubItem = NULL;
   MenuItem[z].NextSelect = NULL;

   MenuText[z].FrontPen = 0; MenuText[z].BackPen = 1;
   MenuText[z].TopEdge = 1;
   MenuText[z].DrawMode = JAM2;
   MenuText[z].ITextFont = NULL;
   MenuText[z].NextText = NULL;
}

MenuItem[0].NextItem = &MenuItem[1];
MenuItem[1].NextItem = &MenuItem[2];
MenuItem[2].NextItem = &MenuItem[3];
MenuItem[3].NextItem = NULL;

MenuText[0].LeftEdge = 15; MenuText[0].IText = (UBYTE *)
"Draw";
MenuText[1].LeftEdge = 20; MenuText[1].IText = (UBYTE *)
"Box";
MenuText[2].LeftEdge = 10; MenuText[2].IText = (UBYTE *)
"Erase";
MenuText[3].LeftEdge = 15; MenuText[3].IText = (UBYTE *)
"Quit";

Menu[0].NextMenu = &Menu[1];
Menu[0].LeftEdge = 10; Menu[0].TopEdge = 0;
Menu[0].Width = 58; Menu[0].Height = 10;
Menu[0].Flags = MENUENABLED;
Menu[0].MenuName = "Actions";
Menu[0].FirstItem = &MenuItem[0];
```

```
 for (z = 4; z < 8; z++)
 {
    MenuItem[z].LeftEdge = 0; MenuItem[z].TopEdge = (z-4)
* 10;
    MenuItem[z].Width = 50; MenuItem[z].Height = 10;
    MenuItem[z].Flags = ITEMENABLED | HIGHBOX;
    MenuItem[z].MutualExclude = NULL;
    MenuItem[z].ItemFill = (APTR) &MenuImage[z-4];
    MenuItem[z].SelectFill = NULL;
    MenuItem[z].Command = NULL;
    MenuItem[z].SubItem = NULL;
    MenuItem[z].NextSelect = NULL;

    MenuImage[z-4].LeftEdge = 1; MenuImage[z-4].TopEdge =
1;
    MenuImage[z-4].Width = 48; MenuImage[z-4].Height = 8;
    MenuImage[z-4].Depth = 2;
    MenuImage[z-4].ImageData = NULL;
    MenuImage[z-4].PlanePick = 0;
    MenuImage[z-4].PlaneOnOff = z-4;
    MenuImage[z-4].NextImage = NULL;
 }

MenuItem[4].NextItem = &MenuItem[5];
MenuItem[5].NextItem = &MenuItem[6];
MenuItem[6].NextItem = &MenuItem[7];
MenuItem[7].NextItem = NULL;

Menu[1].NextMenu = NULL;
Menu[1].LeftEdge = 80; Menu[1].TopEdge = 0;
Menu[1].Width = 44; Menu[1].Height = 10;
Menu[1].Flags = MENUENABLED;
Menu[1].MenuName = "Color";
Menu[1].FirstItem = &MenuItem[4];


SetMenuStrip (Window, &Menu[0]);
```

```
BodyText.FrontPen = 0; BodyText.BackPen = 1;
BodyText.DrawMode = JAM2;
BodyText.LeftEdge = 30; BodyText.TopEdge = 5;
BodyText.ITextFont = NULL;
BodyText.IText = (UBYTE *) "Bist Du sicher?";
BodyText.NextText = NULL;

PositiveText.FrontPen = 0; PositiveText.BackPen = 1;
PositiveText.DrawMode = JAM2;
PositiveText.LeftEdge = 5; PositiveText.TopEdge = 4;
PositiveText.ITextFont = NULL;
PositiveText.IText = (UBYTE *) "Na klar!";
PositiveText.NextText = NULL;

NegativeText.FrontPen = 0; NegativeText.BackPen = 1;
NegativeText.DrawMode = JAM2;
NegativeText.LeftEdge = 5; NegativeText.TopEdge = 4;
NegativeText.ITextFont = NULL;
NegativeText.IText = (UBYTE *) "Hm...";
NegativeText.NextText = NULL;

 SetAPen (Window->RPort, 1);
 SetDrMd (Window->RPort, JAM1);


 while (endflag == TRUE)
 {
    Wait (1 << Window->UserPort->mp_SigBit);

    while (Message = (struct IntuiMessage *) GetMsg (Windo
w->UserPort))
    {
       Class = Message->Class;
       Code = Message->Code;
```

```
        ReplyMsg (Message);

        mousemoveflag = FALSE;

        switch (Class)
        {
           case MOUSEMOVE:
              mousemoveflag = TRUE;
              break;

           case MOUSEBUTTONS:
              switch (Code)
              {
                 case SELECTUP:
                    penflag = UP;
                    break;

                 case SELECTDOWN:
                    penflag = DOWN;
                    Move (Window->RPort, Window->MouseX, Wi
ndow->MouseY);
                    break;

                 case MENUDOWN:

                    SetDrMd (Window->RPort, COMPLEMENT | JA
M1);
                    drawbox (Window, start_x, start_y, end_
x, end_y);
                    SetDrMd (Window->RPort, JAM1);
                    Window->Flags &= (0xffffffff ^ RMBTRAP)
;
                    rubberbandflag = FALSE;
                    break;
              }
              break;

           case  MENUPICK:
              if (Code != MENUNULL)
              {
                 switch (MENUNUM (Code))
                 {
                    case 0:
                       switch (ITEMNUM (Code))
                       {
                          case 0:
                             drawflag = TRUE;
                             boxflag = FALSE;
                             break;

                          case 1:
                             boxflag = TRUE;
                             drawflag = FALSE;
                             break;
```

```
                                  case 2:
                                     if (AutoRequest (Window, &Body
Text, &PositiveText,
                                                         &NegativeText
, NULL, NULL, 200, 50)
                                         == TRUE)
                                           RectFill (Window->RPort,
 0, 0, 320, 200);
                                     break;

                                   case 3:
                                      if (AutoRequest (Window, &Body
 Text, &PositiveText,
                                                          &NegativeText
 , NULL, NULL, 200, 50)
                                          == TRUE)
                                            endflag = FALSE;
                                      break;
                              }
                              break;

                          case 1:
                              SetAPen (Window->RPort, ITEMNUM (Cod
 e));
                              break;
                        }
                     }
                     break;

                  default:
                     break;
               }
            }
```

```
if (boxflag && penflag && !rubberbandflag)
{
   rubberbandflag = TRUE; penflag = FALSE;
   Window->Flags |= RMBTRAP;
   start_x = Window->MouseX; start_y = Window->MouseY;

   end_x = start_x; end_y = start_y;
   SetDrMd (Window->RPort, COMPLEMENT | JAM1);
   drawbox (Window, start_x, start_y, end_x, end_y);
   SetDrMd (Window->RPort, JAM1);
}


if (boxflag && penflag && rubberbandflag)

    {
       rubberbandflag = FALSE; penflag = FALSE;
       Window->Flags &= (0xffffffff ^ RMBTRAP);
       drawbox (Window, start_x, start_y, end_x, end_y);
    }


    if (drawflag && penflag && mousemoveflag)
       Draw (Window->RPort, Window->MouseX, Window->MouseY
);


    if (boxflag && mousemoveflag && rubberbandflag)
    {
       SetDrMd (Window->RPort, COMPLEMENT | JAM1);
       drawbox (Window, start_x, start_y, end_x, end_y);
       end_x = Window->MouseX; end_y = Window->MouseY;
       drawbox (Window, start_x, start_y, end_x, end_y);
       SetDrMd (Window->RPort, JAM1);
    }
 }
```

```
   ClearMenuStrip (Window);
   CloseWindow (Window);
   CloseScreen (Screen);
   CloseLibrary (GfxBase);
   CloseLibrary (IntuitionBase);
}
```

3.3.5.1 Die verschiedenen Funktionen des Malprogrammes:

Haben Sie das Programm zum Laufen gebracht? Ja? Nun, so sehen Sie jetzt einen unberührten blauen Bildschirm vor sich - öde und leer, und förmlich dazu geschaffen, Kreativität wiederzugeben.

Also legen Sie los. Bewegen Sie bei gedrückter linker Maustaste den Mauszeiger über den Bildschirm. Und damit haben Sie auch schon eine der Funktionen kennengelernt: Draw. Drücken Sie nun einmal die rechte Maustaste. In der Titelleiste werden nun hoffentlich zwei Menüpunkte erscheinen. Gehen Sie einmal auf COLOR, und suchen Sie sich eine der Farben aus. Wenn Sie nun wieder auf dem Bildschirm malen, erscheinen die Pixel in der ausgewählten Farbe, z. B. rot statt vorher weiß.

Im linken Menü ACTIONS erhalten Sie vier Untermenüs:

DRAW: Schaltet den oben beschriebenen Draw-Modus ein.

BOX: Wenn Sie diesen Modus eingestellt haben, so können Sie Rechtecke auf dem Bildschirm entwerfen. Drücken Sie die linke Maustaste, so wird der Anfangspunkt gesetzt. Nun können Sie die Maustaste loslassen, und durch Bewegen der Maus das Rechteck auf die gewünschte Größe bringen. Wenn Sie jetzt die linke Maustaste wieder betätigen, wird das Rechteck gemalt.

Wollen Sie das Rechteck nicht auf dem Bildschirm haben, so betätigen Sie einmal kurz die rechte Maustaste.

ERASE: Löscht den Bildschirm mit der aktuellen Zeichenfarbe.

QUIT: Beendet das Programm (natürlich mit Bestätigung).

3.3.5.2 Ein ungefährer Ablauf des Programmes

Zeile 1 - 9:
Einfügen von Headerfiles. Nicht alle werden hier wirklich benötigt, aber wir benutzen den Lattice-C-Compiler, und konnten diese elenden "Warning 61" nicht mehr länger sehen.

Zeile 13 / 14:
UP steht für Zeichencursor oben, und DOWN für Zeichencursor unten (wenn im Drawmodus die linke Maustaste gedrückt wurde).

Zeile 18 / 19:
Und natürlich brauchen wir auch wieder die Library-Base-Pointer.

Zeile 23 - 32:
Ein kleines Unterprogramm zum Zeichnen eines Rechteckes.

Zeile 38 - 57:
Variablendefinition.

Zeile 61 - 76:
Eröffnen der Intuition- und der Grafik-Library.

Zeile 80 - 127:
Eröffnen eines Screens (320 * 200 * 2) und eines Backdrop-Borderless-Windows-ohne-Menüzeile.

Zeile 131 - 202:
Initialisieren der Menüs.

Zeile 206 - 225:
Belegen des notwendigen Krams für einen Autorequester.

Zeile 234:
Beginn der äußersten Schleife.

Zeile 236:
Ist etwas außergewöhnliches passiert? Ja? Dann sendet Intuition eine Nachricht zum Userport.

Zeile 238:
Wenn noch eine Nachricht anliegt, dann wird die Schleife abgearbeitet. Ist keine Nachricht mehr vorhanden, so geht der Ablauf in Zeile 320 weiter.

Zeile 240 / 241:
Werte dienen zum Erkennen der Nachricht, und werden am besten abgespeichert.

Zeile 243:
Löscht die Nachricht, deren Werte eben abgespeichert worden sind.

Zeile 247:
Jetzt wird versucht, die Nachricht zu erkennen. Erstes Erkennungszeichen dabei ist Class. Hier können drei Fälle eingetreten sein:

> MOUSEMOVE:
> Maus wurde bewegt, also entsprechendes Flag setzen.
>
> MOUSEBUTTONS:
> Hier kann es drei weitere Fälle geben, also muß Code zur eindeutigen Identifizierung der Nachricht mit herangezogen werden:
>
> SELECTUP:
> Linke Maustaste nicht gedrückt, also Flag setzen.

SELECTDOWN:
Linke Maustaste gedrückt, also Flag setzen, und Grafikcursor an aktuelle Mausposition setzen.

MENUDOWN:
Rechte Maustaste gedrückt. Wenn gerade eine Box gezogen wurde, so will der Benutzer diese Box nicht auf seinem Schirm haben. Also wird die Box mittels des Mal modus COMPLEMENT gelöscht, und der Rubberband-Modus beendet.

MENUPICK:
Aha, ein Menü wurde ausgewählt. Doch welches? Läßt sich mittels Code bestimmen (und der Macros MENUNUM & ITEMNUM):

Zeile 283:
Draw-Menü ausgewählt, also ent sprechende Flags setzen / löschen.

Zeile 288:
Box-Menü ausgewählt, also entsprechende Flags setzen / löschen.

Zeile 293:
Erase-Menü ausgewählt. Also einen Requester ausgeben, und ggf. das Window löschen.

Zeile 300:
Quit. Auch hier wieder einen Requester ausgeben, und u.U. wirklich das endflag setzen.

Zeile 323 - 332:
Wird ausgeführt, wenn der Benutzer gerade wieder ein neues Rechteck zeichnen will.

Zeile 335 - 340:
Diese Zeilen werden abgearbeitet, wenn der Benutzer sich dazu durchgerungen hat, zum 2. mal die linke Maustaste zu drücken, und so das Rechteck wirklich gezeichnet werden soll.

Zeile 343 / 344:
Diese zwei Zeilen sind für einfaches Zeichnen mittels des Menüpunktes Draw gedacht. Muß nicht viel gemacht werden.

Zeile 347 - 354:
Diese Zeilen behandeln das Rubberbanding. Darauf wird später noch näher eingegangen.

Zeile 360 - 364:
Da das Programm gleich beendet werden soll, ist es am besten, wieder den gesamten belegten Speicher freizugeben. So muß der Speicher, der durch die Verwendung von Menüs belegt wurde, genauso freigegeben werden wie der Speicher für den Screen, das Window sowie die beiden Libraries.

3.3.5.3 Ein paar Worte zu den verwendeten Variablen

**IntuitionBase:* Zeiger auf eine Struktur vom Typ Intuition-Base. Braucht der Amiga, um auf die Intuition-Library zurückgreifen zu können.

**GfxBase:* Wie oben, jedoch für Grafik-Library.

**Screen:* Zeiger auf die Screen-Struktur des verwendeten Screens.

**Window:* Zeiger auf die Window-Struktur des Windows, auf dem das Malprogramm seine Aktionen ausführt.

NewScreen: NewScreen-Struktur, wird zum Eröffnen des Screens gebraucht.

NewWindow: Eine NewWindow-Struktur; wird gebraucht, um die Eigenschaften des verwendeten Windows festzulegen.

MenuItem[8: Ein Feld von acht Variablen vom Typ struct MenuItem. In diesem Feld werden die Eigenschaften der acht Menüpunkte (Draw, Box, Erase, Quit, und die vier Farben) festgelegt.

MenuText[4: Hier kommen die vier Texte zu den Menüpunkten Draw, Box, Erase & Quit hinein. Die vier Farben kommen ohne Text aus.

BodyText: Enthält den Text, der ganz oben im Requester erscheinen soll.

PositiveText: Hier kommt der Text rein, der im Requester in dem linken Gadget erscheinen soll.

NegativeText: Und hier ist der Text für das rechte Gadget im Requester zu finden.

MenuImage[4: Die vier Variablen dieses Feldes beschreiben das Aussehen der vier Menüpunkte, die in dem Menü Color zu finden sind.

Menu[2: Hier werden die beiden Menüs beschrieben.

**Message:* Diese Variable ist ein Zeiger auf die Stelle, wo Intuition eine anfallende Nachricht ablegen soll.

z: Zähler. Wird beim Initialisieren der Menüpunkte gebraucht, da hier ein paarmal dieselbe Information in verschiedenen Variablen abgespeichert werden soll, was am besten mit einer Schleife geschieht.

Class: Wird von Intuition an das Programm gegeben, und enthält eine Information zum Erkennen der Nachricht.

Code: Enthält eine weitere Information zum Erkennen der Nachricht; wird auch von Intuition belegt.

start_x, start_y: Enthalten die Koordinaten des Anfangspunktes des Rechtecks. Diese beiden Variablen werden einmal gesetzt,

und bleiben solange konstant, bis der Benutzer das Rechteck zeichnet, oder das Rubberbanding mittes der rechten Maustaste unterbricht.

end_x, end_y: Diese beiden Variablen enthalten die Koordinaten des Endpunktes des Rechtecks. Beim Rubberbanding enthalten diese Variablen die Position des Mauszeigers, da dieser ja den Endpunkt des aktuellen Rechteckes mit sich herumzieht.

endflag: Dieses Flag ist solange TRUE, bis der Benutzer das Programm mittels des Quit-Menüs beenden will. Durch endflag wird die äußerste while-Schleife kontrolliert.

penflag: Ist UP, wenn der Benutzer die linke Maustaste nicht gedrückt hat, und DOWN, wenn doch.

mousemoveflag: Ist FALSE, wenn der Mauszeiger nicht bewegt wurde. Im anderen Fall TRUE.

drawflag: Gibt an, ob das Draw-Menü ausgewählt wurde. Ist ganz zu Anfang des Programmes mit TRUE besetzt; dadurch können Sie schon von Anfang an loszeichnen.

boxflag: Ist TRUE, wenn der Benutzer Rechtecke zeichnen will, und demnach das Box-Menü ausgesucht hat.

rubberbandflag: Hiermit wird angezeigt, ob das Programm gerade dabei ist, ein bißchen Rubberbanding durchzuführen.

3.3.5.4 Und wie programmiere ich nun Menüs?

Das ist gar nicht so schwer - erfordert jedoch eine ganze Menge Schreibarbeit. Jedes Menü und jedes Untermenü braucht nämlich eine eigene Struktur. Doch schauen wir uns das ganze einmal im Programm an:

Ab Zeile 131 wird das gesamte Menü initialisiert. Jeder Menüpunkt belegt dabei eine Struktur vom Typ MenuItem. Diese Struktur hat folgende Komponenten:

NextItem: Zeigt auf den nächsten Menüpunkt. Dadurch können mehrere Menüpunkte in einer Liste miteinander verbunden werden.

LeftEdge: X-Wert des Punktes, an dem der Menüpunkt anfangen soll.

TopEdge: Y-Wert dieses Punktes.

Width: Länge des Menüpunktes in Bildschirmpixel.

Height: Höhe des Menüpunktes - auch in Bildschirmpixel.

Flags: Hier kann man eine ganze Menge Flags setzen, und so auch eine ganze Menge bestimmen. Einige Flags sind:

ITEMTEXT: Ist dieses Flag gesetzt, so besteht der Inhalt des Menüpunktes aus Text. Ist dieses Flag nicht gesetzt, so ist es ein Bild und wird somit mit einer Image-Struktur beschrieben.

HIGHCOMP: Ist dieses Flag gesetzt, so werden alle Bits innerhalb des Menüpunktes komplementär dargestellt, wenn der Benutzer diesen Menüpunkt auswählt.

HIGHBOX: Zieht ein Rechteck um den Menüpunkt, wenn er angewählt wird.

HIGHIMAGE: Wird ein Menüpunkt, der dieses Flag gesetzt hat, vom Benutzer ausgewählt, so wird innerhalb des Menüpunktes ein völlig neues Image dargestellt.

HIGHNONE: Setzen Sie dieses Flag, falls Sie keine Reaktion auf das Auswählen eines bestimmten Menüpunktes wollen.

ITEMENABLED: Erlaubt dem Benutzer das Anwählen dieses Menüpunktes.

MutualExclude: Oh Gott, dies ist wirklich schwierig in wenigen Worten zu erklären und wird deshalb vorerst ausgelassen.

ItemFill: Hier kommt die Adresse der Struktur hinein, die beschreibt, was innerhalb des Menüpunktes erscheinen soll. Dies könnte z.B. eine IntuiText-Struktur sein, falls Text dargestellt werden soll. Es könnte jedoch auch die Adresse einer Image-Struktur sein. Denken Sie daran, das entsprechende Flag zu setzen.

Beim Malprogramm bestehen die vier Menüpunkte des ersten Menüs (Action) aus Texten, und demnach wird eine IntuiText-Struktur für jeden dieser Menüpunkte verwendet. Die vier Menüpunkte unter Color bestehen aus einfachen Images (ohne jegliches Bild, nur verschiedene Farben), und so wird für diese eine Image-Struktur verwendet.

SelectFill: Ähnlich wie ItemFill, jedoch für eine IntuiText- oder Image-Struktur. Wird in den Menüpunkt eingesetzt, wenn das Flag HIGHIMAGE für diesen Menüpunkt gesetzt wurde.

Command: Wurde das Flag COMMSEQ gesetzt, so enthält diese Komponente ein alphanumerisches Zeichen, das zusammen mit der rechten Amiga-Taste als Abkürzung für diesen Menüpunkt gelten kann.

SubItem: Wenn dieser Menüpunkt irgendwelche Untermenüpunkte besitzen soll, muß diese Variable auf das erste dieser Untermenüpunkte zeigen. Bei einem Untermenüpunkt wird diese Komponente nicht beachtet; man kann also nicht noch weiter verschachteln.

NextSelect: Diese Variable wird von Intuition gefüllt, und gibt an, ob noch weitere Menüpunkte vom Benutzer ausgewählt wurden. Soviel zur MenuItem-Struktur.

Mehrere Menüpunkte werden zu einem Menü mittels der Menu-Struktur zusammengefaßt. Diese Struktur enthält acht Komponenten, und zwar folgende:

NextMenu: Zeigt auf die Menu-Struktur des nächsten Menüs. Mit dieser Komponenten können also mehrere Menüs miteinander verbunden werden.

LeftEdge, TopEdge, Width, Height: Geben Koordinaten und Größe des Menüpunktes an.

Flags: MENUENABLED: Erlaubt dem Benutzer das Auswählen dieses Menüs.

MenuName: Name des Menüs. Achten Sie darauf, daß der Name in das angegebene Feld paßt.

FirstItem: Zeigt auf die MenuItem-Struktur des ersten Menüpunktes für dieses Menü.

Nachdem nun alle Menüpunkte und Menüs miteinander verbunden worden sind, wird die Menüleiste mittels des Intuition-Befehls SetMenuStrip gesetzt. Dieser Befehl benötigt zwei Argumente:

Window: Zeiger auf die Window-Struktur des Windows, für das dieses Menü gesetzt werden soll.

Menu: Zeiger auf die Menu-Struktur des ersten Menüs.

Gelöscht werden kann eine Menüleiste mit dem Intuition-Befehl ClearMenuStrip. Dieser benötigt nur ein Argument: Einen Zeiger auf die Window-Struktur des Windows, dessen Menüs gelöscht werden sollen.

Gut, die Menüleiste ist nun vorhanden, und kann vom Benutzer angewählt werden. Doch wie wird dem Programm übermittelt, welcher Menüpunkt gefragt ist? Werfen Sie dazu einen Blick auf das Listing:

In Zeile 236 wartet das Programm darauf, daß Intuition eine Nachricht übermittelt. Dies wird kenntlich gemacht durch Setzen des Signal-Bits am Userport. Nachdem die Nachricht angekommen ist, wird sie in Zeile 238 abgespeichert. In den Zeilen 240 und 241 werden zwei Werte, die die Nachricht genau beschreiben, in den beiden Variablen *Class* und *Code* abgelegt.

Nun kann die Nachricht am Userport gelöscht werden. Das geschieht mittels ReplyMsg in Zeile 243. Ist die Variable Class gleich MENUPICK, so gibt Intuition bekannt, daß der Benutzer einen Menüpunkt ausgewählt hat. In diesem Fall geht der Programmablauf in Zeile 276 weiter.

Der Macro MENUNUM gibt die Nummer des Menüs, in der der ausgewählte Menüpunkt liegt, an das Programm zurück. Dadurch kann eine erste Unterscheidung getroffen werden. Mit dem Macro ITEMNUM kann nun genau die Nummer des ausgewählten Menüpunktes bestimmt werden.

Ist zum Beispiel die Menünummer = 0 und die Itemnummer = 2, so wurde vom Benutzer der dritte Menüpunkt im ersten Menü ausgewählt (das erste Menü bzw. der erste Menüpunkt hat die Nummer 0 und nicht 1). Also will der Benutzer den Bildschirm löschen (ERASE), und der Programmablauf geht in Zeile 294 weiter.

Um die Nummer von Untermenüpunkten zu bestimmen, existiert auch noch der Macro SUBITEM.

3.3.5.5 Requester ganz automatisch

Das Malprogramm benutzt auch Requester. Keine aufwendigen, grafikunterstützen, mit zahlreichen Gadgets versehene Reque-

ster, sondern einen ganz einfachen. Dieser dürfte jedoch in den meisten Fällen auch ausreichen. Außerdem ist er um vieles einfacher zu programmieren als jeder andere Requester.

Dazu gebraucht man einfach die Intuition-Funktion AutoRequest. Diese Funktion erwartet einige Argumente:

1. Einen Zeiger auf die Window-Struktur des Windows, in dem der Requester erscheinen soll.

2. Einen Zeiger auf die IntuiText-Struktur für den Text, der über den beiden Requester-Gadgets erscheinen soll.

3. Einen Zeiger auf die IntuiText-Struktur des Textes, der in dem linken Gadget erscheinen soll.

4. Und hier der Zeiger für das rechte Gadget.

5. IDCMP-Flags, die, wenn sie auftreten, das gleiche bewirken, wie wenn der Benutzer das linke Gadget angeklickt hat.

6. IDCMP-Flags für das rechte Gadget.

7. Die Breite des Requesters in Bildschirmpunkten.

8. Die Höhe des Requesters in Bildschirmpunkten.

Aufgerufen wird die Funktion AutoRequest innerhalb des Malprogrammes in den Zeilen 294 bzw. 301 - jedesmal mit den gleichen Argumenten.

Bei Aufruf gibt die Funktion AutoRequest einen Wert zurück. Dieser Wert ist TRUE, wenn die Bedingung des linken Gadgets erfüllt wurde, und FALSE, wenn die Antwort dem rechten Gadget entsprach.

Noch ein Tip: Eine sinnvolle Möglichkeit bei dem fünften bzw. sechsten Argument des AutoRequest-Befehls sind die Flags DISKINSERTED bzw. DISKREMOVED. Bei diesen Flags wird der Requester beendet, wenn eine Disk in ein Laufwerk hineingeschoben bzw. herausgeholt wird.

3.3.5.6 Rubberbanding

Ihnen sagt der Begriff "Rubberbanding" überhaupt nichts? Sie werden es nicht glauben, aber Sie arbeiten wahrscheinlich täglich damit! Jedesmal, wenn Sie ein Window vergrößern oder verkleinern, benutzen Sie Rubberbanding.

Unser Malprogramm benutzt Rubberbanding, wenn ein Rechteck gemalt werden soll. Solange der Benutzer noch nicht zum zweiten Mal den linken Mausknopf gedrückt hat, kann er auf dem Bildschirm hin- und herfahren, und zieht dabei vier zusammenhängende Linien munter hin und her. Diese Linien übermalen jedoch nichts. Diesen Effekt nennt man Rubberbanding. Rubberbanding könnte auch auftreten, wenn man mit einem Malprogramm, zum Beispiel Graphicraft, Linien zeichnen will. Oder Kreise.

Beim Rubberbanding benutzt man den Drawmode COMPLEMENT. Dieser Modus sorgt dafür, daß beim Zeichnen nicht irgendwelche absoluten Farben benutzt werden; stattdessen wird einfach die Hintergrundfarbe komplementiert. So wird aus %01 = Farbregister 1 das Farbregister %10 = 2. Und aus %10111 wird %01000. Weiterhin nutzt Rubberbanding aus, daß man eine Linie, die im COMPLEMENT-Modus gezeichnet wurde, ganz einfach wieder löschen kann, indem man - immer noch im Zeichen-Modus COMPLEMENT - die gleiche Linie noch einmal zieht. So wird aus %10100 der Wert %01011, und schließlich wieder %10100.

In dem Programms sieht das so aus:

Drückt der Benutzer zum ersten mal den linken Mausknopf, so wird in den Zeilen 323 bis 332 ein Rechteck im COMPLE-

MENT-Modus an der aktuellen Mausposition gezeichnet. Außerdem wird mittels des Flags RMBTRAP dafür gesorgt, daß das Drücken der rechten Maustaste zu einer ganz normalen MOUSEBUTTONS-Nachricht führt.

Fährt der Benutzer nun mit der Maus hin und her, so gerät das Programm in die Zeilen 347 bis 354. Hier wird zuerst mit den alten Rechteck-Koordinaten noch ein Rechteck gezogen. Dadurch wird das alte Rechteck gelöscht. Danach wird die aktuelle Mausposition als neuer Endpunkt des Rechteckes gesetzt, und mit diesen neuen Werten wieder ein Rechteck gezogen. Dieses erscheint nun jedoch auf dem Bildschirm, und wird erst beim nächsten mal wieder gelöscht.

Ist der Benutzer schließlich der Meinung, daß das Rechteck an der richtigen Position ist, so drückt er noch einmal die linke Maustaste, und landet in Zeile 335. Hier wird der Rubberband-Modus beendet, die rechte Maustaste wieder auf den normalen Stand gebracht, und schließlich das gewünschte Rechteck gezeichnet - diesmal jedoch nicht im COMPLEMENT-Modus, sondern mit der ganz normalen Zeichenfarbe.

Ist der Benutzer der Meinung, das Rechteck, daß er gerade im Rubberband-Modus mit sich rumzieht, beginnt eigentlich am falschen Punkt, oder gehört überhaupt nicht auf den Bildschirm, so braucht er nur die rechte Maustaste zu drücken. In diesem Fall geht der Programmablauf in Zeile 265 weiter. Hier wird mittels des COMPLEMENT das zuletzt gezeichnete Rechteck gelöscht. Dann wird die rechte Maustaste wieder normalisiert, und schließlich durch Rücksetzen der Flags der Rubberband-Modus beendet.

3.3.5.7 Ein Schlußwort

Das Malprogramm ist natürlich nur ein grobes Gerüst. Es fehlen noch Menüpunkte zum Laden, Speichern und Ausdrucken der Bilder. Doch beginnt man erst einmal damit, dann braucht man auch noch Kreise, 32 Farben, und Interlace. Also haben wir hier

aufgehört. Da das Programm jedoch einigermaßen übersichtlich und durchstrukturiert geschrieben wurde, dürfte es nicht weiter schwierig sein, entsprechende Menüpunkte einzubauen.

3.3.6 Und noch ein Grafik-Modus: Half-Brite

Sie haben gelesen, daß der Amiga bei seiner niedrigsten Auflösung von 320 * 200 Bildschirmpunkte 32 Farben darstellen kann, und waren total begeistert? Was halten Sie dann davon, wenn wir Ihnen sagen, daß der Amiga bei dieser Auflösung 64 verschiedene Farben darstellen kann? 64 mögliche Farben für jeden Bildschirmpunkt!

Wollen Sie auch? Gut, dann benutzen Sie doch in Zukunft den Half-Brite-Modus. Der Half-Brite-Modus wird eingeschaltet, indem man beim Eröffnen eines neuen Screens in der Screenstruktur die Komponente Depth auf 6, und die Komponente ViewModes mit dem Flag EXTRA_HALFBRITE belegt.

Und hier das obligatorische Demo-Programm:

```
#include <exec/types.h>
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


main ()
{
   struct Screen *Screen;
   struct Window *Window;
   struct NewScreen NewScreen;
   struct NewWindow NewWindow;
   COUNT zaehler;
```

```
 IntuitionBase = (struct IntuitionBase *)
                     OpenLibrary ("intuition.library", 0)
;
 if (IntuitionBase == NULL)
 {
    printf ("Help if You can I`m feeling down...\n");
    exit (FALSE);
 }

 GfxBase = (struct GfxBase *) OpenLibrary ("graphics.libra
ry", 0);
 if (GfxBase == NULL)
 {
    printf ("Help! I need somebody...\n");
    CloseLibrary (IntuitionBase);
    exit (FALSE);
 }


NewScreen.LeftEdge = 0; NewScreen.TopEdge = 0;
NewScreen.Width = 320; NewScreen.Height = 200;
NewScreen.Depth = 6;
NewScreen.DetailPen = 0; NewScreen.BlockPen = 1;
NewScreen.ViewModes = EXTRA_HALFBRITE;
NewScreen.Type = CUSTOMSCREEN;
NewScreen.Font = NULL;
NewScreen.DefaultTitle = NULL;
NewScreen.Gadgets = NULL;
NewScreen.CustomBitMap = NULL;

Screen = (struct Screen *) OpenScreen (&NewScreen);
if (Screen == NULL)
{
   printf ("Sorry, no Screen...\n");
   CloseLibrary (IntuitionBase);
   CloseLibrary (GfxBase);
   exit (FALSE);
}
```

```
NewWindow.LeftEdge = 0; NewWindow.TopEdge = 0;
NewWindow.Width = 320; NewWindow.Height = 200;
NewWindow.DetailPen = 0; NewWindow.BlockPen = 1;
NewWindow.IDCMPFlags = NULL;
NewWindow.Flags = SMART_REFRESH | BACKDROP | BORDERLESS |
ACTIVATE |
                  NOCAREREFRESH;
NewWindow.FirstGadget = NULL;
NewWindow.CheckMark = NULL;
NewWindow.Title = NULL;
NewWindow.Screen = Screen;
NewWindow.BitMap = NULL;
NewWindow.MinWidth = NULL; NewWindow.MinHeight = NULL;
NewWindow.MaxWidth = NULL; NewWindow.MaxHeight = NULL;
NewWindow.Type = CUSTOMSCREEN;

Window = (struct Window *) OpenWindow (&NewWindow);
if (Window == NULL)
{

   printf ("Sorry, no Window...\n");
   CloseLibrary (IntuitionBase);
   CloseLibrary (GfxBase);
   CloseScreen (Screen);
   exit (FALSE);
}


ShowTitle (Screen, FALSE);
```

```
   for (zaehler = 0; zaehler < 64; zaehler++)
   {
      SetAPen (Window->RPort, zaehler);
      RectFill (Window->RPort, zaehler * 10, 0, (zaehler * 1
  0) + 4, 200);
      SetAPen (Window->RPort, zaehler + 32);
      RectFill (Window->RPort, (zaehler * 10) + 5, 0, (zaehl
  er * 10) + 9, 200);
   }


   while ((Window->MouseX != 0) || (Window->MouseY != 0));


   CloseWindow (Window);
   CloseScreen (Screen);
   CloseLibrary (GfxBase);
   CloseLibrary (IntuitionBase);
}
```

In Zeile 41 wird die Anzahl der Bitplanes mit 6 angegeben, und in Zeile 43 wird nicht INTERLACE, HIRES sondern EXTRA_HALFBRITE gesetzt. Danach wird wieder das Backdrop-Borderless-Window eröffnet, und die Titelzeile gelöscht.

Und in Zeilen 91-97 werden nun 64 Streifen in den Farben der 64 Farbregister auf den Bildschirm gebracht. Dabei wird zuerst eines der Farbregister 0 - 31 benutzt und sofort rechts daneben ein Rechteck in der Farbe des Farbregisters, welches 32 Nummern höher liegt, gemalt.

Wenn Sie das Programm nun gestartet haben, bewundern Sie die Vielfalt. Doch schauen Sie nun etwas genauer hin. Fällt Ihnen etwas auf?

O.K., wir beichten: In Wirklichkeit können zwar 64 verschiedene Farben dargestellt werden, jedoch nicht 64 verschiedene. Die Inhalte der oberen 32 Farbregister wird durch die Inhalte der unteren 32 bestimmt:

Es gehören immer zwei Register zusammen: 0 und 32, 1 und 33, 2 und 34, usw. Das höhere Farbregister enthält dabei die Rot-,

Blau- und Grün-Anteile der Farbe des niedrigeren Registers, alle ein Bit nach rechts geschoben. So wird aus dem Grünanteil %1010 = 10 der Grünanteil %0101 = 5.

Schauen Sie nun noch einmal genau die 64 Rechtecke an. Der rechte Balken ist immer von der gleichen Farbe wie der linke - jedoch doppelt so dunkel (oder auch halb so hell; daher auch der Name Half-Brite).

Wichtig ist beim Halfbrite-Modus ein geschicktes Belegen der Farbregister, aus zwei verschiedenen Farbregisterinhalten kann durchaus die gleiche Half-Brite-Farbe werden. So ergeben sowohl %1101 als auch %1100 die Halfbrite-Farbe %0110, da das Bit 0 einfach rausgeschoben wird, und somit keine Bedeutung hat.

Ach ja, beenden kann man das Demoprogramm, indem man den Mauszeiger in die linke obere Ecke des Bildschirms bringt (Zeile 101).

3.3.7 Interlace - soweit das Auge blickt

Der Amiga kennt zahlreiche verschiedene Grafikauflösungen. Zuerst einmal gibt es da die Grundauflösung von 320 * 200 Bildschirmpixel. Gibt man beim Eröffnen eines Screens bei der Komponente ViewModes das Flag HIRES an, so erhält man ein Auflösung von 640 * 200 Bildschirmpixel, d.h. die horizontale Auflösung wird verdoppelt. Gibt man dagegen das Flag INTERLACE an, so verdoppelt sich die vertikale Auflösung von 200 Bildschirmpixel. Kombiniert man die Flags HIRES und INTERLACE so ergibt sich die maximale (naja, nahezu maximale...) Bildschirmauflösung von 640 * 400 Pixel.

Beim Interlace-Modus wird das Bild in zwei Teilbilder aufgeteilt, die um eine Bildschirmzeile versetzt immer abwechselnd auf den Bildschirm gebracht werden. Diesen Trick mußte man sich einfallen lassen, da 400 Bildschirmzeilen nicht mehr einfach so hintereinander dargestellt werden können.

Leider flackert der Bildschirm im Interlace-Modus etwas (bzw. etwas mehr), da die effektive Bildschirm-Frequenz halbiert wurde, und so das menschliche Auge schon die einzelnen Bilder unterscheiden kann.

Doch zahlreiche Grafikanwendungen kommen ohne die Auflösung von 640 * 400 Pixel nicht aus, also muß trotz dieses starken Nachteiles auf den Interlace-Modus zurückgegriffen werden.

Das Flackern läßt sich übrigens stark herabsetzen bzw. ganz ausschalten, wenn man einerseits die Regler für Helligkeit bzw. Kontrast am Monitor etwas verändert, sowie andererseits die verwendeten Bildschirmfarben mit Hilfe von Preferences etwas nachstellt (so sollte man aus dem Weiß ein helles Grau machen). Am besten solange rumprobieren, bis ein annehmbares Bild erreicht ist.

Hier nun ein kleines Programm zum Interlace-Modus. Das fertig compilierte und gelinkte Programm wird folgendermaßen aufgerufen (angenommen, Sie haben das Programm wie wir "Interlace" genannt):

"Interlace on": Schaltet Interlace-Modus für alle momentan offenen Screens ein.

"Interlace off": Schaltet den Interlace-Modus wie der aus.

So, und hier nun das Listing:

```
#include <graphics/gfxbase.h>
#include <intuition/intuition.h>


struct GfxBase *GfxBase;
struct IntuitionBase *IntuitionBase;
```

```
main (argc, argv)
   int argc;
   char *argv[];
{
   BOOL InterlaceON;



   if (argc != 2)
   {
      printf ("Ein Argument: `on` oder `off`\n");
      exit (FALSE);
   }

   if (! strcmp (argv[1], "on"))
      InterlaceON = TRUE;
   else if (! strcmp (argv[1], "off"))
      InterlaceON = FALSE;
   else
   {
      printf ("??? Entweder `on` oder `off`!!!\n");
      exit (FALSE);
   }



   GfxBase = (struct GfxBase *) OpenLibrary ("graphics.library", 0);
   if (GfxBase == NULL)
   {
      printf ("Mal wieder keine Graphik-Bibliothek...\n");
      exit (FALSE);
   }

   IntuitionBase = (struct IntuitionBase *)
                        OpenLibrary ("intuition.library", 0);
   if (IntuitionBase == NULL)


   {
      printf ("Diesmal fehlt die Intuition-Library...\n");
      CloseLibrary (GfxBase);
      exit (FALSE);
   }
```

```
    if (InterlaceON)
       GfxBase->system_bplcon0 |= INTERLACE;
    else
       GfxBase->system_bplcon0 &= !INTERLACE;


    RemakeDisplay ();


    CloseLibrary (IntuitionBase);
    CloseLibrary (GfxBase);
}
```

Vielleicht sollten wir auch noch ein paar Erklärungen abgeben:

- Die Variable "InterlaceON" ist ein Flag und kann demnach zwei Zustände annehmen: TRUE, wenn der Interlace-Modus eingeschaltet, und FALSE, falls er wieder ausgeschaltet werden soll.

- In den Zeilen 20-34 werden die Argumente, die der Benutzer hinter dem Programmnamen angegeben hat, ausgewertet. Zuerst einmal wird geschaut, ob überhaupt die richtige Anzahl (eins!) von Argumenten vorliegt. Ist dies nicht der Fall, so wird in Zeile 22 eine entsprechende Meldung ausgegeben, und das Programm beendet. Sodann wird geprüft, ob das Argument "on" angegeben wurde. Wenn ja, so wird das Flag InterlaceON für spätere Verwendung auf TRUE gesetzt. Wenn nein, so wird weiter getestet: Wurde das Argument "off" angegeben? Wenn ja, so wird wieder das Flag InterlaceON für spätere Verwendung gesetzt - diesmal auf FALSE. Wenn nein, so wird eine Meldung für den Benutzer ausgegeben, da er ein falsches Argument angegeben hat.

- Danach werden die Intuition- und die Grafik-Bibliothek eröffnet. Hat irgendetwas dabei nicht geklappt, so wird eine entsprechende Meldung ausgegeben, alles wieder geschlossen, was vorher irgendwann mal eröffnet wurde, und dann das Programm beendet.

- Und dann kommen schließlich die alles entscheidenden Zeilen: 57-60. In Zeile 57 wird das Flag InterlaceON abgefragt. Ist es gesetzt (TRUE), so wird die Komponente system_bplcon0 mit dem Macro INTERLACE logisch oder verknüpft, d.h. das Flag INTERLACE wird gesetzt. Ist InterlaceON auf FALSE gesetzt, so werden die einzelnen Bits des Flags INTERLACE innerhalb der Komponenten system_bplcon0 mittels einer logsichen Und-Verknüpfung gelöscht.

- Warum eigentlich ausgerechnet die Komponente system_bplcon0? Diese Komponente hat eine besondere Funktion. Um diese Funktion zu erklären, müssen wir etwas weiter ausholen: Die gesamte Grafik wird beim Amiga nicht durch den 68000 erledigt, sondern durch den Copper, einen Koprozessoren. Dieser Koprozessor verfügt über eine eigene Maschinensprache. Alles, was Sie auf dem Bildschirm sehen, seien es Screens, Windows, Gadgets oder Grafiken, wird in diese Maschinensprache übersetzt. Das dabei entstehende Programm wird dann vom Copper ausgeführt; der Copper führt daraufhin bestimmt Aktionen aus, und läßt so das auf dem Monitor zu sehende Bild erscheinen. Die Maschinensprache-Listings greifen ziemlich oft auf ein bestimmtes Hardware-Register zurück. Dieses Hardware-Register nennt sich bplcon0; es enthält die möglichen Darstellungsmodi wie HIRES, SPRITES - und auch INTERLACE.

Die Komponente system_bplcon0 innerhalb der GfxBase-Struktur steht nun - wie unschwer am Namen zu erkennen - in enger Verbindung zu diesem Hardware-Register. Jedesmal wenn ein Copper-Maschinensprache-Listing auf dieses Hardware-Register zugreift, wird diese Komponente logisch oder mit diesem Hardware-Register verknüpft. Dadurch werden automatisch alle Flags,

die in der Komponente gesetzt sind, auch in dem Hardware-Register gesetzt. Setzen wir nun in der Komponente das Flag INTERLACE, so wird auch in allen Copper-Listings das Flag INTERLACE gesetzt. Und so haben auf einmal alle Screens den Interlace-Modus eingeschaltet.

- In Zeile 64 wird schließlich mit dem Intuition-Befehl RemakeDisplay dafür gesorgt, daß alle Maschinensprache-Listings, die den Copper dazu veranlassen, das aktuelle Bild darzustellen, neu gebildet werden. Beim neuen Bilden wird die Komponente system_bplcon0 mit dem Hardware-Register bplcon0 oder-verknüpft, und so stellen alle Listings jetzt - ob vorher gewollt oder nicht - den Interlace-Modus an.

Wird ein Screen, der vorher nicht im Interlace-Modus war, auf einmal in den Interlace-Modus umgeschaltet, so ergibt sich dennoch kein allgemeines Chaos - es wird einfach jede Pixelzeile verdoppelt. Im Gegenteil - läßt sich das Flackern abschalten, so ist das Bild nun besser als vorher, da die schmalen Leerzeilen zwischen den einzelnen Bildschirmzeilen verschwunden sind, und man so nicht mehr die einzelnen Punkte bei Buchstaben und Grafiken erkennen kann.

3.4 Devices

Die Programmierer des Amiga haben versucht, das System möglichst ausbaufähig und dennoch einfach zu halten. So hat man auch bei den Devices versucht, einen gewissen Standard zu erreichen, damit für den Programmierer zwischen den einzelnen Devices kaum ein Unterschied besteht.

So hat jedes Device eine Sprungtabelle mit Standardfunktionen wie READ, WRITE, RESET, etc. Zusätzlich hat jedes Device noch einige Extra-Befehle, zum Beispiel FORMAT beim Trackdisk-Device.

3.4.1 "Etwas Theorie" oder "Wie eröffne ich ein Device?"

Da die Devices für den Programmierer alle ziemlich gleich aussehen, kann man auch für alle so ziemlich die gleiche Programmschritte verwenden:

1. Definieren eines Zeigers auf eine Struktur vom Typ MsgPort:

   ```
   struct MsgPort *deviceport;
   ```

2. Definieren eines Zeigers auf eine Struktur vom Typ IOStdReq:

   ```
   struct IOStdReq *devicemsg;
   ```

3. Eröffnen eines Messageports und auf Fehler checken:

   ```
   deviceport = (struct MsgPort *)
   CreatePort (#name, #priorität);
   if (deviceport == NULL) errorhandling;
   ```

4. Initialisieren des Requests & auf Fehler checken:

   ```
   devicemsg = (struct IOStdReq *)
   CreateStdIO (deviceport);
   if (devicemsg == NULL) errorhandling;
   ```

5. Eröffnen des Devices & auf Fehler checken:

   ```
   if (OpenDevice (#devicename, #unit, devicemsg,
   #flags) != 0) errorhandling;
   ```

6. Gewünschten Befehl in den Request eintragen:

```
device->io_Command = #command;
device->io_Actual = #data;
device->io_Length = #data;
device->io_Data = #data;
...
```

7. Senden der Nachricht zum Device. Hier gibt es zwei Möglichkeiten:

   DoIO (devicemsg): Schickt die Nachricht ab, und wartet, bis das Device den Befehl ausgeführt hat.

   SendIO (devicemsg): Schickt die Nachricht ab, und kommt sofort wieder zurück. Dadurch können noch andere Aktionen ausgeführt werden, während das Device arbeitet. Um zu testen, ob das Device inzwischen fertig ist, benutzt man den Befehl CheckIO (devicemsg). Ist das Device noch nicht fertig, so gibt CheckIO den Wert 0 zurück. Ist das Device fertig, so bekommt der Aufrufer einen Zeiger auf den Request.

8. Schließen des Devices, des Requests und des Ports:

```
CloseDevice (devicemsg);
DeleteStdIO (devicemsg);
DeletePort (deviceport);
```

Abhängig vom verwendeten Device kann es zu einigen Änderungen dieses Ablaufs kommen.

3.4.2 Maus verändern mit dem Input-Device

Das Input-Device verfügt über zwei recht interessante Befehle: SETMPORT und SETMTYPE. Mit SETMPORT kann bestimmt

werden, an welchem Port das Gerät ist, das den Mauszeiger kontrolliert. Mit SETMTYPE wird die Art dieses Gerätes festgelegt.

Kontrollieren Sie jedoch vorher, ob Sie (wie wir) eine alte Version des Headerfiles input.h besitzen. Bei dieser Version sind die Befehle SETMPORT und SETMTYPE nicht implementiert. Das Headerfile sollte ungefähr so aussehen:

```
#ifndef   DEVICES_INPUT_H
#define   DEVICES_INPUT_H
/***********************************************************
  **********/
/*                 Commodore-Amiga Inc.
           */
/*                 input.h
           */
/***********************************************************
  **********/
/***********************************************************
  ***********
*
*   input device command definitions
*
************************************************************
  **********/
#ifndef      EXEC_IO_H
#include   "exec/io.h"
#endif

#define   IND_ADDHANDLER       (CMD_NONSTD+0)
#define   IND_REMHANDLER       (CMD_NONSTD+1)
#define   IND_WRITEEVENT       (CMD_NONSTD+2)
#define   IND_SETTHRESH        (CMD_NONSTD+3)
#define   IND_SETPERIOD        (CMD_NONSTD+4)
#define   IND_SETMPORT         (CMD_NONSTD+5)
#define   IND_SETMTYPE         (CMD_NONSTD+6)
#define   IND_SETMTRIG         (CMD_NONSTD+7)

#endif
```

Zuerst einmal ein Programm zu dem Befehl SETMTYPE:

```
#include <exec/types.h>
#include <exec/devices.h>
#include <devices/input.h>


main (argc, argv)
   int argc;
   char *argv[];
{
   struct MsgPort *deviceport;
   struct IOStdReq *devicemsg;
   UBYTE mouseport;


   if (argc != 2)
   {
      printf ("Bad args\n");
      exit (FALSE);
   }


   switch (argv[1][0])
   {
      case '?':
         printf ("Ein Argument: Nr. des Ports fuer die Maus
  (0 / 1)\n");
         exit (TRUE);

      case '0':
         mouseport = 0;
         break;

      case '1':
         mouseport = 1;
         break;

      default:
         printf ("Bad args: 0 or 1\n");
```

```
      exit (FALSE);
}


deviceport = (struct MsgPort *) CreatePort (0, 0);
if (deviceport == NULL)
{
   printf ("Sorry, no DevicePort\n");
   exit (FALSE);
}

devicemsg = (struct IOStdReq *) CreateStdIO (deviceport);

if (devicemsg == NULL)
{
   printf ("Sorry, no DeviceMsg\n");
   DeletePort (deviceport);
   exit (FALSE);
}

if (OpenDevice ("input.device", 0, devicemsg, 0) != 0)
{
   printf ("Sorry, no InputDevice\n");
   DeletePort (deviceport);
   DeleteStdIO (devicemsg);
   exit (FALSE);
}


devicemsg->io_Command = IND_SETMPORT;
devicemsg->io_Data = (APTR) &mouseport;
devicemsg->io_Length = 1;


DoIO (devicemsg);
```

```
    CloseDevice (devicemsg);
    DeleteStdIO (devicemsg);
    DeletePort (deviceport);
}
```

Bei Aufruf dieses Programmes müssen Sie ein Argument übergeben. Dieses Argument kann entweder 0 oder 1 sein, je nachdem, an welchem Port die Maus anliegen soll. In den Zeilen 24 bis 41 wird getestet, ob die richtigen Argumente vorliegen. Ist dies nicht der Fall, so wird das Programm mit einer entsprechenden Mitteilung beendet.

Danach werden der Reihe nach ein Port, ein Standard- Request, und danach das Input-Device eröffnet.

In den Zeilen 72 bis 74 werden dann die notwendigen Variablen zum Befehl SETMPORT gesetzt. Dazu muß in die Komponente io_Command innerhalb der Device-Message das Flag IND_SETMPORT gesetzt werden. In io_Data kommt ein Zeiger auf das Byte, das angibt, ob der Mausport 0 oder 1 gesetzt werden soll. Und schließlich muß Length noch mit 1 belegt werden.

Danach kann der Befehl an das Input-Device gesendet werden. Da sonst nichts weiter geschehen soll, wird dieses mit dem Befehl DoIO vollbracht.

Nachdem alles gut abgelaufen ist, wird das Device, der Request und der Port wieder geschlossen, und das Programm beendet.

Nun ein Programm zu dem zweiten Befehl:

```
#include <exec/types.h>
#include <exec/devices.h>
#include <devices/input.h>
#include <devices/gameport.h>


main (argc, argv)
   int argc;
   char *argv[];
{
   struct MsgPort *deviceport;
   struct IOStdReq *devicemsg;
   UBYTE mousetype;


   if (argc != 2)
   {
      printf ("Ich brauche 1 Argument...\n");
      exit (FALSE);
   }


   switch (argv[1][0])
   {
      case '?':
         printf ("Art der Maus: m = Maus, j = Joystick\n");
         exit (TRUE);

      case 'm':
         mousetype = GPCT_MOUSE;
         break;

      case 'j':
         mousetype = GPCT_RELJOYSTICK;
         break;

      default:
         printf ("Falscher Aufruf\n");
```

```
      exit (FALSE);
}


deviceport = (struct MsgPort *) CreatePort (0, 0);
if (deviceport == NULL)
{
   printf ("Sorry, no DevicePort\n");
   exit (FALSE);
}


devicemsg = (struct IOStdReq *) CreateStdIO (deviceport);

if (devicemsg == NULL)
{
   printf ("Sorry, no DeviceMsg\n");
   DeletePort (deviceport);
   exit (FALSE);
}


if (OpenDevice ("input.device", 0, devicemsg, 0) != 0)
{
   printf ("Sorry, no InputDevice\n");
   DeletePort (deviceport);
   DeleteStdIO (devicemsg);
   exit (FALSE);
}


devicemsg->io_Command = IND_SETMTYPE;
devicemsg->io_Data = (APTR) &mousetype;
devicemsg->io_Length = 1;


DoIO (devicemsg);

   CloseDevice (devicemsg);
   DeleteStdIO (devicemsg);
   DeletePort (deviceport);
}
```

Das Programm ist dem obrigen absolut ähnlich. Diesmal heißt der Befehl jedoch IND_SETMTYPE, und in devicemsg->io_Data wird ein Flag übergeben. Laut dem Headerfile devices/gameport.h gibt es hierbei fünf Möglichkeiten:

GPCT_ALLOCATED: Port ist schon benutzt.

GPCT_NOCONTROLLER: Kein Controller mehr. Die Builder Maussteuerung per Tastatur geht allerdings immer noch, da die Tastatur nicht hierüber beeinflußt wird.

GPCT_MOUSE: Steuerung des Mauszeigers durch die ganz normale Maus.

GPCT_RELJOYSTICK: Joystick kontrolliert Mauszeiger.

GPCT_ABSJOYSTICK: Vermutlich Trackball - da wir jedoch über keinen verfügen, konnte das nicht nachgeprüft werden.

Bei Aufruf erwartet das Programm wieder ein Argument:

m: schaltet auf Maus um.
j: schaltet auf Joystick um.

3.4.3 Zugriff auf den Drucker

Auch der Drucker ist ein Device, und deshalb genauso leicht anzusprechen wie die Maus. Trotzdem ist unser Beispielprogramm um einiges länger. Dies liegt daran, daß noch einige zusätzliche Funktionen aufgeführt werden müssen, die für die Programmierung des Druckers unabdingbar sind. Diese Funktionen bringen Sie am besten in einer Bibliothek unter; binden Sie diese Bibliothek dann zu jedem Programm, das auf den Drucker zugreift.

Hier nun ein kleines Beispielprogramm, das den aktuellen Screen auf den Drucker ausgibt:

```
#include "exec/types.h"
#include "exec/exec.h"
#include "intuition/intuition.h"
#include "devices/printer.h"


extern APTR AllocMem();


union printerIO
   {
      struct IOStdReq ios;
      struct IODRPReq iodrp;
      struct IOPrtCmdReq iopc;
   };



struct IORequest *CreateExtIO (ioReplyPort, size)
   struct MsgPort *ioReplyPort;
   LONG size;
{
   struct IORequest *ioReq;


   if (ioReplyPort == 0)
      return ((struct IORequest *) 0);


   ioReq = (struct IORequest *) AllocMem (size, MEMF_CLEAR |
   MEMF_PUBLIC);
   if (ioReq == 0)
      return ((struct IORequest *) 0);


   ioReq->io_Message.mn_Node.ln_Type = NT_MESSAGE;
   ioReq->io_Message.mn_Node.ln_Pri = 0;
   ioReq->io_Message.mn_ReplyPort = ioReplyPort;
```

```
   return (ioReq);
}


DeleteExtIO (ioExt, size)
   struct IORequest *ioExt;
   LONG size;
{
   ioExt->io_Message.mn_Node.ln_Type = 0xff;
   ioExt->io_Device = (struct Device *) -1;
   ioExt->io_Unit = (struct Unit *) -1;

   FreeMem (ioExt, size);
}


int OpenPrinter (request)
   union printerIO *request;
{
   return (OpenDevice ("printer.device", 0, request, 0));
}


ClosePrinter (request)
   union printerIO *request;
{
   CloseDevice (request);
}


int DumpRPort (request, rastPort, colorMap, modes, sx, sy, s
  w, sh, dc, dr, s)
   union printerIO *request;
```

```
   struct RastPort *rastPort;
   struct ColorMap *colorMap;
   ULONG modes;
   UWORD sx, sy, sw, sh;
   LONG dc, dr;
   UWORD s;
{
   request->iodrp.io_Command = PRD_DUMPRPORT;
   request->iodrp.io_RastPort = rastPort;
   request->iodrp.io_ColorMap = colorMap;
   request->iodrp.io_Modes = modes;
   request->iodrp.io_SrcX = sx;
   request->iodrp.io_SrcY = sy;
   request->iodrp.io_SrcWidth = sw;
   request->iodrp.io_SrcHeight = sh;
   request->iodrp.io_DestCols = dc;
   request->iodrp.io_DestRows = dr;
   request->iodrp.io_Special = s;

   return (DoIO (request));
}


union printerIO *request;

struct IntuitionBase *IntuitionBase;


main ()
{
   struct RastPort *rastport;
   struct ColorMap *colormap;
   int modes, width, height, error;
   struct Port *printerPort;
```

```
 IntuitionBase = (struct IntuitionBase *)
                      OpenLibrary ("intuition.library", 0)
;
 if (IntuitionBase == NULL)
 {
    printf ("Intuition-Library laesst sich nicht oeffnen!!
!\n");
    exit (FALSE);
 }


 rastport = &(IntuitionBase->ActiveScreen->RastPort);

 colormap = IntuitionBase->ActiveScreen->ViewPort.ColorMap
;

 modes = IntuitionBase->ActiveScreen->ViewPort.Modes;

 width = IntuitionBase->ActiveScreen->ViewPort.DWidth;
 height = IntuitionBase->ActiveScreen->ViewPort.DHeight;


 printerPort = (struct Port *) CreatePort (0, 0);
 if (printerPort == NULL)
 {
    printf ("Ich bekomme keinen Port!!!\n");
    CloseLibrary (IntuitionBase);
    exit (FALSE);
 }


 request = (union printerIO *)
                CreateExtIO (printerPort, sizeof (union pr
interIO));
 if (request == NULL)
 {
```

```
        printf ("CreateExtIO klappt nicht!!!\n");
        DeletePort (printerPort);
        CloseLibrary (IntuitionBase);
        exit (FALSE);
    }


    if (OpenPrinter (request) != NULL)
    {
        printf ("Printer laesst sich nich oeffnen!!!\n");
        DeleteExtIO (request, sizeof (union printerIO));
        DeletePort (printerPort);
        CloseLibrary (IntuitionBase);
        exit (FALSE);
    }



    error = DumpRPort
                (
                    request, rastport, colormap, modes,
                    0, 0, width, height, width, (height * 2),
                    SPECIAL_FULLROWS | SPECIAL_FULLCOLS | SPEC
   IAL_ASPECT
                );

    if (error != NULL)
        printf ("Hm, irgendwas hat nicht ganz geklappt...\n");




    ClosePrinter (request);
    DeleteExtIO (request, sizeof (union printerIO));
    DeletePort (printerPort);
    CloseLibrary (IntuitionBase);
}
```

Das Programm besteht aus sechs Funktionen:

CreateExtIO: Ähnelt der Funktion CreateStdIO; ist jedoch für Devices gedacht, die sich nicht mit den

gewöhnlichen Request-Strukturen zufriedengeben - wie z.B. der Printer.

DeleteExtIO: Gegenstück zu DeleteStdIO.

OpenPrinter: Öffnet den Drucker.

ClosePrinter: Schließt den Drucker wieder.

DumpRPort: Gibt einen Rastport auf den Drucker aus.

main: Unser eigentliches Programm. Macht nicht viel mehr als alle oberen Routinen der Reihe nach aufzurufen.

Kommen wir nun zu den einzelnen Funktionen:

1. CreateExtIO:

Vergleichen wir CreateExtIO einmal mit der Funktion CreateStdIO:

```
struct IOStdReq *CreateStdIO (ioReplyPort)
   struct MsgPort *ioReplyPort;
{
   struct IoStdReq *ioStdReq;

   if (ioReplyPort == 0)
      return ((struct IoStdReq *) 0);

   ioStdReq = AllocMem (sizeof (*ioStdReq), MEMF_CLEAR | MEMF_PUBLIC);
   if (ioStdReq == 0)
      return ((struct IoStdReq *) 0);

   ioStdReq->io_Message.mn_Node.ln_Type = NT_MESSAGE;
   ioStdReq->io_Message.mn_Node.ln_Pri = 0;
   ioStdReq->io_Message.mn_ReplyPort = ioReplyPort;

   return (ioStdReq);
}
```

Es existiert nur ein Unterschied: Die IO soll sich nicht auf die Standard-Request-Struktur (IOStdReq) beziehen, sondern auf eine individuelle. Aus diesem Grund muß bei CreateExtIO die Größe der jeweiligen Request-Struktur mit als Argument übergeben werden. Ansonsten erfüllen beide Funktionen die gleichen Aufgaben: Sie reservieren den benötigten Speicherplatz auf einer 4-Byte-Grenze, und belegen sogleich noch einige wichtige Komponenten innerhalb der reservierten Struktur; unter anderem wird so ein Messageport angegeben, über den das Programm mit dem Device kommunizieren kann.

2. DeleteExtIO:

Auch hier wieder ein Vergleich mit der normalen Funktion DeleteStdIO:

```
DeleteSTdIO (ioStdReq)
   struct ioStdReq *ioStdReq;
{
   ioStdReq->io_Message.mn_Nodce.oln_Type = 0xff;
   ioStdReq->io_Device = (struct Device *)-1;
   ioStdReq->io_Unit = (struct Unit *) -1;


   Freemem (ioStdReq, sizeof (*ioStdReq);
}
```

Auch bei diesen beiden Funktionen liegt der einzige Unterschied darin, daß die eine auf die normale Request-Struktur zurückgreift, während die andere eine individuelle Struktur verwaltet. Aus diesem Grund kann die eine von einem festen Menge belegten Speicherplatzes ausgehen, während die andere die Anzahl der belegten Bytes als Argument übergeben muß.

Ansonsten fabrizieren beide Funktionen wieder dasgleiche: die Devices, Units, Speicher und dergleichen wird wieder freigegeben.

3. OpenPrinter:

Dürfte ohne weitere Erklärungen verständlich sein.

4. ClosePrinter:

Nähere Erklärungen siehe 3.

5. DumpRPort:

Hm, jetzt wird es schwieriger mit dem Erklären. Das Printer-Device verfügt über einen sehr leistungsfähigen Befehl: PRD_DUMPRPORT. Sendet man diesen Befehl an das Printer-Device, so versucht dieses, den angegebenen Rastport auf den Drucker auszugeben. Dazu kann man eine ganze Menge Angaben machen, die den Ausdruck stark variieren können:

io_RastPort: Zeiger auf die RastPort-Struktur des auszugebenen Rastportes. Je nachdem, ob man den Rastport eines Screens, eines Windows, oder eines Sonstwas nimmt, kann man einen Screen, ein Window, oder ein Sonstwas auf den Drucker ausgeben.

io_ColorMap: Zeiger auf ColorMap-Struktur.

io_Modes: Enthält die Darstellungsmodi für den auszugebenen Grafikteil (z. B. INTERLACE, HIRES, SPRITES).

io_SrcX: X-Offset in den Rastport.

io_SrcY: Y-Offset in den Rastport.

io_SrcWidth: Weite des auszugebenen Teiles des Rastports.

io_SrcHeight: Höhe dieses Teiles.

io_DestCols: Zusammen mit io_DestRows und io_Special wird mit dieser Komponenten bestimmt, in welchem Format der auszugebene Rastport auf dem Printer erscheinen soll.

io_DestRows: siehe io_DestCols.

io_Special: Hier gibt es zur Zeit folgende Möglichkeiten (an die Makros muß immer noch COLS bzw. ROWS angehangen werden, je nachdem, ob sich das Flag auf Spalten oder Zeilen beziehen soll):

SPECIAL_MIL: Ist dieses Flag gesetzt, werden die Komponenten io_DestCols / io_DestRows als Tausenstel eines Inches interpretiert.

SPECIAL_FULL: Der Rastport wird so groß wie überhaupt möglich ausgegeben - das heißt, so groß, wie es Preferences bzw. der Drucker erlaubt.

SPECIAL_FRAC: ???

SPECIAL_ASPECT: Ist dieses Flag gesetzt, so kann u.U. vom Printer-Device eine Ausdehnung etwas verändert werden, damit ein wirklichkeitsgetreuer Ausdruck erfolgt.

SPECIAL_DENSITYMASK: ???

SPECIAL_DENSITY1: ???

SPECIAL_DENSITY2: ???

SPECIAL_DENSITY3: ???

SPECIAL_DENSITY4: ???

Zu den letzten fünf Flags erbitten wir uns Zuschriften von Leuten, die mehr darüber wissen!

Die Funktion DumpRPort macht nun nichts weiter, als alle diese Komponenten innerhalb des Requests mit den aufgerufenen Argumenten zu belegen, und sodenn den Request an das Printer-Device abzuschicken.

Durch die Funktion in Zeile DoIO wartet DumpRPort solange, bis sich das Device wieder meldet.

Hat irgendwas nicht ganz geklappt, so gibt DoIO - und damit auch DumpRPort - den Wert Null zurück.

6. main:

Innerhalb von main wird zuerst einmal wie gehabt die Intuition-Library eröffnet. Tritt hier schon ein Fehler auf, so wird das Programm schleunigst mit einer entsprechenden Mitteilung verlassen.

Sodann werden in den Zeilen 131 - 138 die Variablen rastport, colormap, modes, width und height mit den entsprechenden Werten aus der Screen-Struktur des gerade aktiven Screens geholt. Diese Variablen werden später an beim Aufruf der Funktion DumpRPort an dieselbige übergeben. Dadurch wird der aktuelle Screen auf den Drucker ausgegeben.

Danach wird ein Messageport eröffnet. Über diesen Port soll nachher mit dem Printer-Device Signal-Austausch erfolgen. Gleich dahinter wird der Request initialisiert - mittels der am Anfang des Programmes aufgeführten CreateExtIO-Funktion.

Sollte dies alles fehlerlos vonstatten gegangen sein, so schließlich in der Zeile 173 die Funktion DumpRPort aufgerufen. Wir möchten gerne den Drucker voll ausnutzen, und dennoch ein wirklichkeitsgetreues Bild haben - also setzen wir die Flags SPECIAL_FULLROWS, SPECIAL_FULLCOLS und SPECIAL_ASPECT. Danach wird wieder alles geschlossen, und das Programm ist fertig.

Das Printer-Device verfügt noch über weitere Befehle außer PRT_DUMPRPORT. Diese führen wir jetzt hier auf - komplett mit den Komponenten, die für diese Befehle belegt werden müssen.

> FLUSH: Standard-Device-Befehl; veranlaßt das Device, alle IO augenblicklich abzubrechen.

> PRD_PRTCOMMAND: Mittels dieses Befehles kann ein Befehl an den Drucker gesendet werden. Schauen Sie sich dazu folgende kleine Routine an:

```
int PrintCommand (request, command, p0, p1, p2, p3)
   union printerIO *request;
   int command, p0, p1, p2, p3;
{
   request->iopc.io_Command = PRD_PRTCOMMAND;
   request->iopc.io_PrtCommand = command;
   request->iopc.io_Parm0 = p0;
   request->iopc.io_Parm1 = p1;
   request->iopc.io_Parm2 = p2;
   request->iopc.io_Parm3 = p3;

   return (DoIO (request));
}
```

Mittels dieser kleinen Routinen können Befehle zum Drucker gesandt werden. Ein Beispiel für den Aufruf dieser Routine:

```
PrintCommand (request, aSGR4, 0, 0, 0, 0);
```

Dieser Aufruf schaltet das automatisch Unterstreichen an (sofern der Drucker über diese Option verfügt). Der Befehl aSGR4 stammt aus dem Include-File devices/printer.h und entspricht der Escape-Sequence ESC + "[4m". Da das Include-File devices/printer.h mit diesen Befehlen nur so gespickt ist, drucken wir es hier ab:

```
#ifndef     DEVICES_PRINTER_H
#define     DEVICES_PRINTER_H
/****************************************************************************/
/*               Commodore-Amiga Inc.                                       */
/*               printer.h                                                  */
/****************************************************************************/
/****************************************************************************
 *
 *   printer device command definitions
 *
 *  Source Control
 *  --------------
 *  $Header: printer.h,v 1.2 85/10/09 16:16:10 kodiak Exp $
 *
 *  $Locker:  $
 *
 ****************************************************************************/

#ifndef   EXEC_NODES_H
#include   "exec/nodes.h"
#endif

#ifndef   EXEC_LISTS_H
#include   "exec/lists.h"
#endif

#ifndef   EXEC_PORTS_H
#include   "exec/ports.h"
#endif

#define   PRD_RAWWRITE      (CMD_NONSTD+0)
#define   PRD_PRTCOMMAND    (CMD_NONSTD+1)
#define   PRD_DUMPRPORT     (CMD_NONSTD+2)

/* printer command definitions */

#define aRIS      0  /* ESCc  reset                       ISO */
#define aRIN      1  /* ESC#1 initialize                  +++ */
#define aIND      2  /* ESCD  lf                          ISO */

#define aNEL      3  /* ESCE  return,lf                   ISO */
#define aRI       4  /* ESCM  reverse lf                  ISO */
```

```
#define aSGR0    5   /* ESC[0m normal char set            ISO */
#define aSGR3    6   /* ESC[3m italics on                 ISO */
#define aSGR23   7   /* ESC[23m italics off               ISO */
#define aSGR4    8   /* ESC[4m underline on               ISO */
#define aSGR24   9   /* ESC[24m underline off             ISO */
#define aSGR1   10   /* ESC[1m boldface on                ISO */
#define aSGR22  11   /* ESC[22m boldface off              ISO */
#define aSFC    12   /* SGR30-39  set foreground color    ISO */
#define aSBC    13   /* SGR40-49  set background color    ISO */

#define aSHORP0 14   /* ESC[0w normal pitch               DEC */
#define aSHORP2 15   /* ESC[2w elite on                   DEC */
#define aSHORP1 16   /* ESC[1w elite off                  DEC */
#define aSHORP4 17   /* ESC[4w condensed fine on          DEC */
#define aSHORP3 18   /* ESC[3w condensed off              DEC */
#define aSHORP6 19   /* ESC[6w enlarged on                DEC */
#define aSHORP5 20   /* ESC[5w enlarged off               DEC */

#define aDEN6   21   /* ESC[6"z shadow print on           DEC (sort of) *
/
#define aDEN5   22   /* ESC[5"z shadow print off          DEC */
#define aDEN4   23   /* ESC[4"z doublestrike on           DEC */
#define aDEN3   24   /* ESC[3"z doublestrike off          DEC */
#define aDEN2   25   /* ESC[2"z  NLQ on                   DEC */
#define aDEN1   26   /* ESC[1"z  NLQ off                  DEC */

#define aSUS2   27   /* ESC[2v superscript on             +++ */
#define aSUS1   28   /* ESC[1v superscript off            +++ */
#define aSUS4   29   /* ESC[4v subscript on               +++ */
#define aSUS3   30   /* ESC[3v subscript off              +++ */
#define aSUS0   31   /* ESC[0v normalize the line         +++ */
#define aPLU    32   /* ESCL  partial line up             ISO */
#define aPLD    33   /* ESCK  partial line down           ISO */

#define aFNT0   34   /* ESC(B US char set                 DEC */
#define aFNT1   35   /* ESC(R French char set             DEC */
#define aFNT2   36   /* ESC(K German char set             DEC */
#define aFNT3   37   /* ESC(A UK char set                 DEC */
#define aFNT4   38   /* ESC(E Danish I char set           DEC*/
#define aFNT5   39   /* ESC(H Sweden char set             DEC*/
#define aFNT6   40   /* ESC(Y Italian char set            DEC */
#define aFNT7   41   /* ESC(Z Spanish char set            DEC */
#define aFNT8   42   /* ESC(J Japanese char set           +++ */
#define aFNT9   43   /* ESC(6 Norweign char set           DEC */
#define aFNT10  44   /* ESC(C Danish II char set          +++ */

#define aPROP2  45   /* ESC[2p  proportional on           +++ */
#define aPROP1  46   /* ESC[1p  proportional off          +++ */
#define aPROP0  47   /* ESC[0p  proportional clear        +++ */
#define aTSS    48   /* ESC[n E set proportional offset   ISO */
#define aJFY5   49   /* ESC[5 F auto left justify         ISO */
```

```
#define aJFY7    50  /* ESC[7 F auto right justify       ISO */
#define aJFY6    51  /* ESC[6 F auto full justify        ISO */
#define aJFY0    52  /* ESC[0 F auto justify off         ISO */
#define aJFY3    53  /* ESC[3 F letter space (justify)   ISO (special) *
/
#define aJFY1    54  /* ESC[1 F word fill(auto center)   ISO (special) *
/

#define aVERP0   55  /* ESC[0z  1/8" line spacing        +++ */
#define aVERP1   56  /* ESC[1z  1/6" line spacing        +++ */
#define aSLPP    57  /* ESC[nt  set form length n        DEC */
#define aPERF    58  /* ESC[nq  perf skip n (n>0)        +++ */
#define aPERF0   59  /* ESC[0q  perf skip off            +++ */

#define aLMS     60  /* ESC#9   Left margin set          +++ */
#define aRMS     61  /* ESC#0   Right margin set         +++ */
#define aTMS     62  /* ESC#8   Top margin set           +++ */
#define aBMS     63  /* ESC#2   Bottom marg set          +++ */
#define aSTBM    64  /* ESC[Pn1;Pn2r  T&B margins        DEC */
#define aSLRM    65  /* ESC[Pn1;Pn2s  L&R margin         DEC */
#define aCAM     66  /* ESC#3   Clear margins            +++ */

#define aHTS     67  /* ESCH    Set horiz tab            ISO */
#define aVTS     68  /* ESCJ    Set vertical tabs        ISO */
#define aTBC0    69  /* ESC[0g  Clr horiz tab            ISO */
#define aTBC3    70  /* ESC[3g  Clear all h tab          ISO */
#define aTBC1    71  /* ESC[1g  Clr vertical tabs        ISO */
#define aTBC4    72  /* ESC[4g  Clr all v tabs           ISO */
#define aTBCALL  73  /* ESC#4   Clr all h & v tabs       +++ */
#define aTBSALL  74  /* ESC#5   Set default tabs         +++ */
#define aEXTEND  75  /* ESC[Pn"x extended commands       +++ */

struct IOPrtCmdReq {
    struct  Message io_Message;
    struct  Device  *io_Device;       /* device node pointer  */
    struct  Unit    *io_Unit;         /* unit (driver private)*/
    UWORD   io_Command;               /* device command */
    UBYTE   io_Flags;
    BYTE    io_Error;                 /* error or warning num */
    UWORD   io_PrtCommand;            /* printer command */
    UBYTE   io_Parm0;                 /* first command parameter */
    UBYTE   io_Parm1;                 /* second command parameter */
    UBYTE   io_Parm2;                 /* third command parameter */
    UBYTE   io_Parm3;                 /* fourth command parameter */
};

struct IODRPReq {
    struct  Message io_Message;
    struct  Device  *io_Device;       /* device node pointer  */
    struct  Unit    *io_Unit;         /* unit (driver private)*/
    UWORD   io_Command;               /* device command */
    UBYTE   io_Flags;
```

```
    BYTE    io_Error;                   /* error or warning num */
    struct  RastPort *io_RastPort;      /* raster port */
    struct  ColorMap *io_ColorMap;      /* color map */
    ULONG   io_Modes;                   /* graphics viewport modes */
    UWORD   io_SrcX;                    /* source x origin */
    UWORD   io_SrcY;                    /* source y origin */
    UWORD   io_SrcWidth;                /* source x width */
    UWORD   io_SrcHeight;               /* source x height */
    LONG    io_DestCols;                /* destination x width */
    LONG    io_DestRows;                /* destination y height */
    UWORD   io_Special;                 /* option flags */
};

#define SPECIAL_MILCOLS    0x001        /* DestCols specified in 1/1000"
*/
#define SPECIAL_MILROWS    0x002        /* DestRows specified in 1/1000"
*/
#define SPECIAL_FULLCOLS   0x004        /* make DestCols maximum possible
 */
#define SPECIAL_FULLROWS   0x008        /* make DestRows maximum possible
 */
#define SPECIAL_FRACCOLS   0x010        /* DestCols is fraction of FULLCO
LS */
#define SPECIAL_FRACROWS   0x020        /* DestRows is fraction of FULLRO
WS */
#define SPECIAL_ASPECT     0x080        /* ensure correct aspect ratio */
#define SPECIAL_DENSITYMASK  0xf00      /* masks out density bits */
#define SPECIAL_DENSITY1   0x100        /* lowest res */
#define SPECIAL_DENSITY2   0x200        /* next res */
#define SPECIAL_DENSITY3   0x300        /* next res */
#define SPECIAL_DENSITY4   0x400        /* hightes res */

#define PDERR_CANCEL          1         /* user canceled a printer timeou
t */
#define PDERR_NOTGRAPHICS     2         /* printer cannot output graphics
 */
#define PDERR_INVERTHAM       3         /* cannot invert hold & modify pr
int */
#define PDERR_BADDIMENSION    4         /* print dimensions illegal */
#define PDERR_DIMENSIONOVFLOW    5      /* print dimensions too large */
#define PDERR_INTERNALMEMORY 6          /* no memory for internal variabl
es */
#define PDERR_BUFFERMEMORY    7         /* no memory for print buffer */
#endif
```

PRT_RAWWRITE: Schickt Daten zum Drucker. Dabei enthält die Komponente io_Length innerhalb der IOStdReq-Struktur innerhalb der printerIO-Union die Anzahl der Daten, und io_Data die zu sendenden Daten.

RESET: Standard-Befehl. Setzt das Device in den Anfangs-Zustand.

START: Standard-Befehl. Startet das durch den Standard-Befehl STOP angehaltene Device wieder.

STOP: Standard-Befehl. Setzt ein Device in den Pause-Zustand. Das Device reagiert jetzt nur noch auf die Befehle START, FLUSH und RESET.

AmigaBASIC für alle. Im ersten Teil werden Sie Schritt für Schritt – und vor allem auf verständliche Art und Weise – in die Programmierung des AMIGA eingeführt: Grafik und Sound gehören genauso dazu wie Datenverwaltung und Statistik. Im zweiten Teil finden Sie alle gelernten Befehle mit Syntax- und Parameterangaben zum schnellen Nachschlagen. Dazu gibt es Programme und Utilities in Hülle und Fülle.



Aus dem Inhalt:

- Das Videotitel-Programm zeigt die OBJECT-Animation
- Das Balken- und Tortengrafik-Programm erklärt die Grafikbefehle
- Das Malprogramm mit Windows, Pulldowns, Mausbefehlen, Füllmustern, Einlesen und Abspeichern von IFF-Bildern
- Das Statistikdaten-Programm hilft, sequentielle Dateien zu verstehen
- Die Datenbank zeigt den Umgang mit relativen Dateien
- Das Sprachutility sorgt für mehr Verständnis bei der Sprachprogrammierung
- Das Synthesizer-Programm führt Sie in die Welt der Töne, Wellenformen und Hüllkurven

Rügheimer, Spanik
AmigaBASIC
Hardcover, 775 Seiten, DM 59,–
ISBN 3-89011-209-9

DAS STEHT DRIN:

Amiga Tips und Tricks ist eine riesige Fundgrube für den Amiga-Besitzer. Viele Beispielprogramme in BASIC und C zeigen, wie man die fantastischen Möglichkeiten dieses Superrechners optimal nutzen kann. Und ganz nebenbei lernt man noch eine Menge über den Aufbau des Computers und seine Programmierung.

Aus dem Inhalt:

- Nutzung der wichtigsten Libraries von BASIC aus: Graphic, DOS, Exec, Intuition
- Nutzung der verschiedenen Disk-Fonts in BASIC-Programmen
- Verschiedene Schrifttypen in BASIC-Programmen: Bold, Outline, Shadow
- Zugriff auf das CLI von BASIC aus
- Bewegbare Screens und Windows mit eigenen Titeln
- Intuition in eigenen Programmen nutzen: Autorequest, Guru Meditation
- Gesamte Directory-Struktur ausdrucken
- Ein-/Ausgabehandling: Diskmonitor, Hardcopy von Windows und Screen
- Speicherverwaltung: AllocMem und FreeMem
- Filehandling in C: Anzahl freier Blöcke, File exist Prüfung, Filegröße, Filekommentar, Get Protection Prüfung
- Zugriff auf Intuition am Beispiel eines einfachen Grafikprogramms: Screen, Windows, Menue
- Half Bright und Interlace Modus
- Druckerhandling in C

UND GESCHRIEBEN HABEN DIESES BUCH:

Tobias Weltner und Ralf Hornig sind Erfolgsautoren der DATA BECKER Bücher ‚64 Tips und Tricks Band 2' und ‚128 Tips und Tricks'. Während eines längeren Aufenthalts in den USA haben sich beide schon frühzeitig mit dem Amiga beschäftigen können.

ISBN 3-89011-211-0